Spring 2013

Parallel-connected solar arrays

Majed Meshal Alabbass

Department:

PARALLEL-CONNECTED SOLAR ARRAYS

by

MAJED MESHAL ALABBASS

A THESIS

Presented to the Faculty of the Graduate School of the

MISSOURI UNIVERSITY OF SCIENCE AND TECHNOLOGY

In Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

2013

Approved by

Dr. J. W. Kimball, Advisor
Dr. M. Ferdowsi
Dr. S. Baur

ABSTRACT

The purpose of this thesis is to investigate the advantages of having various series-parallel configurations of solar arrays and make a comparison between them. The objective is to determine the best PV system configuration and thus improve the overall efficiency of a solar array. The primary focus of this thesis is to study the parallel connection of PV panels for achieving maximum efficiency while extracting maximum energy from the solar radiation. A comparison between a series connected and a parallel connected solar PV array justifies the need for installing a parallel configured solar PV array to achieve optimum performance. The DC-DC converter plays an important role in delivering maximum power to the load. A solar sensor array was used to monitor the solar radiation under various climatic conditions. Data saved using these sensors was then analyzed using software developed with MATLAB's Graphical User Interface (GUI) platform. The PV-cell equations cannot be solved with the ordinary numerical method due to both the complexity and their non-linearity. These calculation were simplified through using the Newton-Raphson (NR) method along with other numerical approximation approaches. The software package is capable of displaying a number of curves including the I-V characteristic, the output power, and the output energy of the PV-panels for different configurations. Various scenarios were simulated and compared under different climatic conditions. The proposed method for parallel configured PV panel was found to be an alternative to existing methods.

ACKNOWLEDGEMENTS

First and foremost, it is with immense gratitude that I acknowledge the support and help of my research advisor, Dr. Kimball. Dr. Kimball continually conveyed a spirit of adventure with regard to both my research and motivation. He inspired and encouraged me to my work on this project. His guidance has helped me throughout this project. I would not have been able to complete this project successfully without his support.

Besides my advisor, I sincerely thank the members of the supervisory committee, Dr. Baur and Dr. Ferdowsi, for their encouragement and insightful comments. They gave me the moral support and the freedom I needed to move on.

Moreover, I am deeply grateful to the Saudi Arabian Cultural Mission (SACM) for the research assistantship they provided. I must also thank the National Science Foundation for supporting this work under award ECCS-0900940. Finally, I want to express my sincerest and deepest gratitude to AL Jouf University for awarding me with a full scholarship to pursue my degree.

Last but not the least, I would like to thank my family and friends. Special thanks to the spirit of my precious mother and to my father for his countless efforts, support, guidance, motivation, and inspiration. His patient love and encouragement enabled me to complete this project. His unconditional support, both financially and emotionally, throughout my degree has been greatly appreciated.

TABLE OF CONTENTS

Page

LIST OF ILUSTRATIONS

Page

LIST OF TABLES

Page

1. INTRODUCTION

Nature possesses several renewable energy resources. Of these renewable resources, solar energy is considered to be the most pure form of energy, as it can be utilized without producing any hazardous waste. The amount of solar energy available is highly dependent on the geographic locations. Some regions receive high solar radiation while others receive only partial radiation. Solar energy however has now been used everywhere to meet both industrial and domestic needs. The amount of solar radiation per unit area of earth is approximately 1000 W/m^2 [1]. This amount of energy is quite high. Thus, many researchers are working diligently to find the most efficient method possible to utilize this clean energy source.

Statistics [2] show that due to an increased interest in solar energy, by many developed countries, the number of installed solar facilities has increased over the last five years. Germany, Spain, Japan, and the United States are all producing a large amount of energy from solar resources. Germany is producing 9783 MW from solar resources followed by Spain and Japan which are producing 3386MW and 2633 MW respectively. United States is producing around 1650 MW utilizing solar radiations.

In 2011, the renewable power consumption grew to 17.7% worldwide. This energy produced approximately 3.9% of the world's electricity. The production of solar power grew 73% in 2011, more than 10 times that of the previous five years combined [3]. Through various foreign investments, Saudi Arabia is currently working to increase

its solar power through various projects in the desert areas where there is no shortfall of sunshine.

Between 1990 and 2009, the number of solar manufacturers increased rapidly in the United States; the number of solar panels shipped grew from 3645 to 10,511 [2]. These statistics indicate how quickly the world is shifting from fossil fuels to renewable energy, especially solar energy. The mobilization of solar power has increased rapidly as well. International manufactures that were supplying solar panels are decreasing as the number of locally manufactured solar panels is increasing.

Though renewable energy is developed rapidly, it still needs to be further researched and developed. When compared to conventional methods, solar power still lacks both reliability and efficiency. This work studied several PV configurations to determine the best PV system configuration and thus improve the efficiency of solar arrays. Different PV configurations were simulated using MATLAB. The data used was measured with sensors and saved in an Excel file.

1.1. LITERATURE REVIEW

1.1.1 PV Model. The solar cell is based on various semiconductor materials. These materials convert solar energy into electricity, also known as the photovoltaic effect. A 10 × 10 cm silicon solar cell under sunlight will produce approximately 0.5V. Multiple solar cells are connected in a series to form a PV module, thus increase this voltage output [4]. Multiple PV modules are connected together to build both PV panels

and PV arrays (a connection of multiple panels). These arrays can be placed on rooftops for power generation.

Various studies have been conducted to identify the best PV cell model available [5-7]. According to previous scholarly work, the most common models used are the single diode and the double diode [5]. The single diode contains a current source in parallel to a diode. Both series and shunt resistance were added to model the loss mechanism inside the PV cell. The equivalent circuit is illustrated in Figure 1.1.

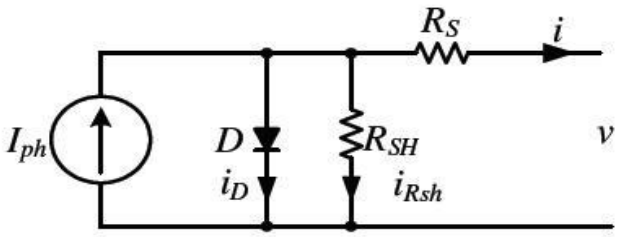


Figure 1.1 Equivalent Circuit of a PV Cell

This equivalent circuit depends on five main parameters: I_{ph}, I_o, R_s, R_{SH}, and A, where I_{ph} is the photo generated current, I_o is the dark saturation current, R_s is the panel series resistance, R_{SH} is the panel parallel (shunt) resistance, and A is the diode quality (ideality) factor.

The single exponential model based current-voltage characteristic of a PV panel is

$$i = I_{ph} - I_o(e^{\frac{v+iR_s}{n_s V_t}} - 1) - \frac{v+iR_s}{R_{sh}} \quad (1)$$

where $V_t = \frac{AkT_{STC}}{q}$ is the junction thermal voltage, k is Boltzmann's constant, q is the electron charge, and n_s is the number of cells in the panel connected in a series.

Five equations must be solved to determine the five unknown parameters.

When $i = I_{SC}$ and $v = 0$,

$$I_{SC} = I_{ph} - I_o e^{\frac{I_{sc}R_s}{n_s v_t}} - \frac{I_{sc}R_s}{R_{sh}} \tag{2}$$

When $i = I_{\max}$ and $v = V_{mpp}$,

$$I_{mpp} = I_{ph} - I_o e^{\frac{V_{mpp}+I_{mpp}R_s}{n_s v_t}} - \frac{v_{mpp}+I_{mpp}R_s}{R_{sh}} \tag{3}$$

When $v = V_{oc}$ and $i = 0$,

$$I_{oc} = 0 = I_{ph} - I_o e^{\frac{v_{oc}}{n_s v_t}} - \frac{V_{oc}}{R_{sh}} \tag{4}$$

The derivative of power with voltage equal to zero at MPP when $i = I_{mpp}$ and $v = V_{mpp}$,

$$\frac{dP}{dV} = 0 \tag{5}$$

The derivative of the current with voltage (when $i = I_{SC}$) can be calculated using shunt resistance R_{Sh}:

$$\frac{dI}{dV} = -\frac{1}{R_{Sh}} \tag{6}$$

Five PV parameters can be determined by solving the above five equations, i.e., equations (2-6). Equation (1) must be solved to find the PV characteristic curves. This equation is non-linear and hence complex to solve using ordinary numerical method. As a result, the Newton-Raphson algorithm is proposed to find the PV characteristic curves.

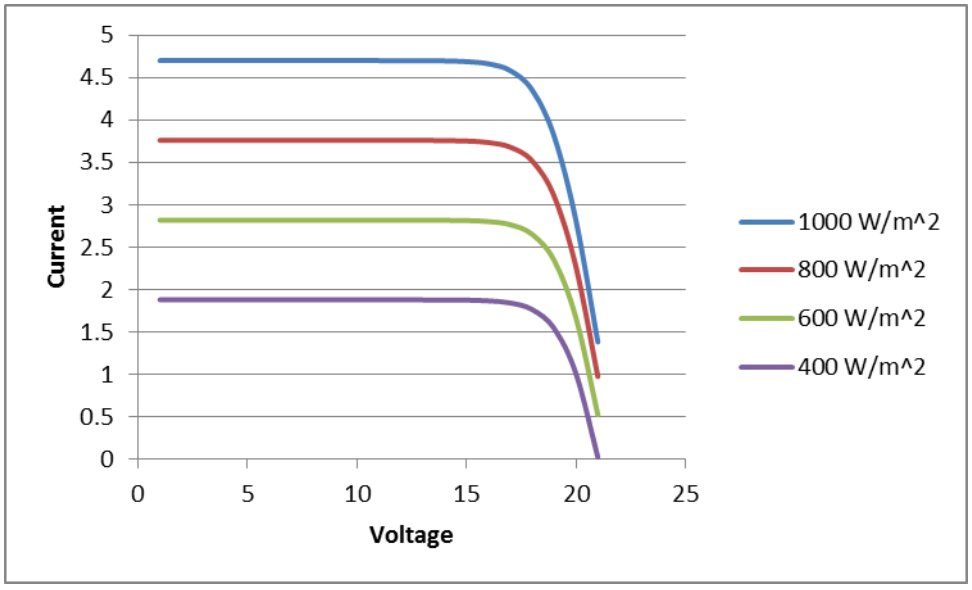
Many researchers have explained the effect of shading in a PV system [8, 9]. Gao and Dougal [8] explained the parallel-connected PV-system under a shadow condition. The output of an entire string within a conventional series configuration will be affected by shading one module. If no bypass diodes exist, the output from the entire series-connected panel will be reduced. If the bypass diodes are connected, the output of the shaded module will be lost. Reducing the output energy in one string will cause difficulty while tracking the Maximum Power Point (MPP) due to the existence of multiple local MPP.

1.1.2 The Effect of Changing the PV Model Parameters. The existing literature discusses the effect of changing the PV cell parameters and checks the output of PV cell in different insolation levels [7] as below:

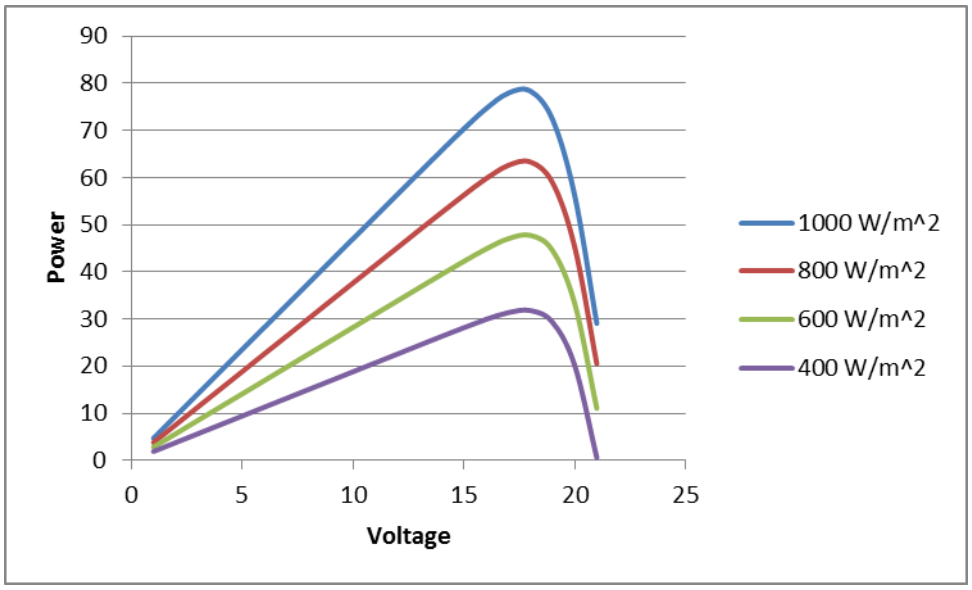
- According to above equation (1), when the photo-generated current decreases (which depends on the solar radiation), the output current will also decrease.
- Figure 1.2 illustrates that the PV system is strongly dependent on solar radiation.
- Figure 1.3 illustrates that the shunt resistor (R_{Sh}) needs to be large for more output power.

- Figure 1.4 illustrates the effects of the series resistance (R_S) variation.

  This resistance needs to be low for a higher output to be produced.

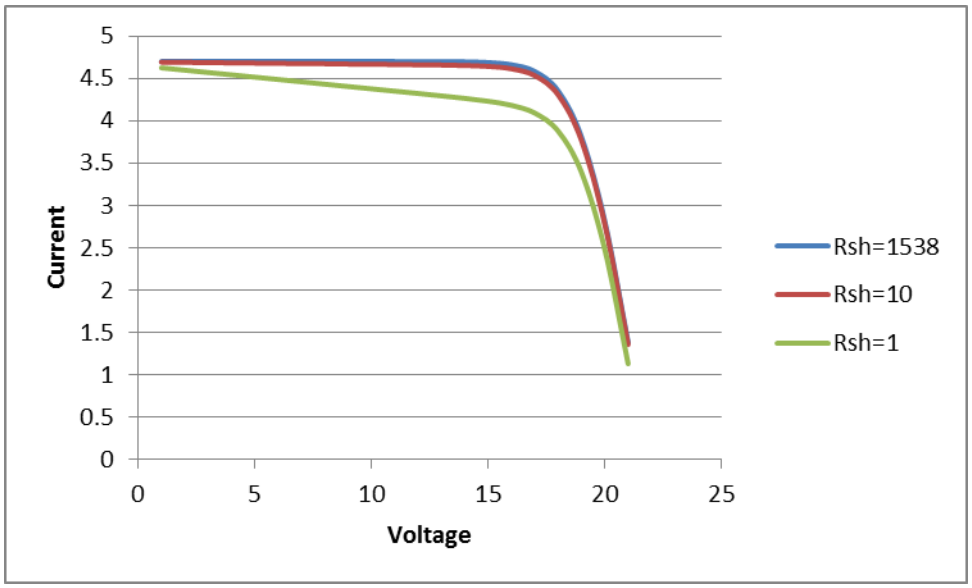


(a) Current-Voltage Curves

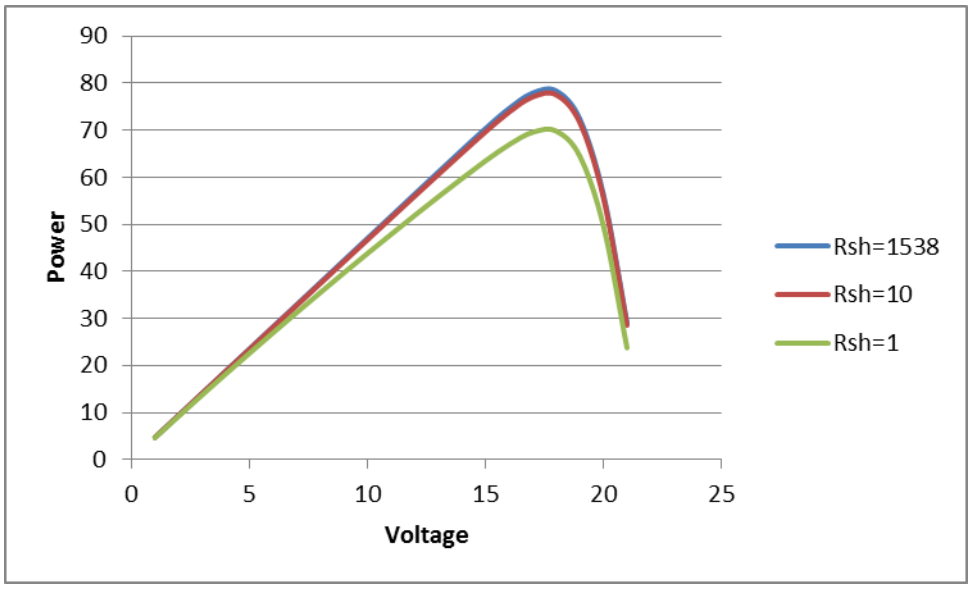


(b) Power-Voltage Curves

Figure 1.2 Effects of Solar Radiation Variation

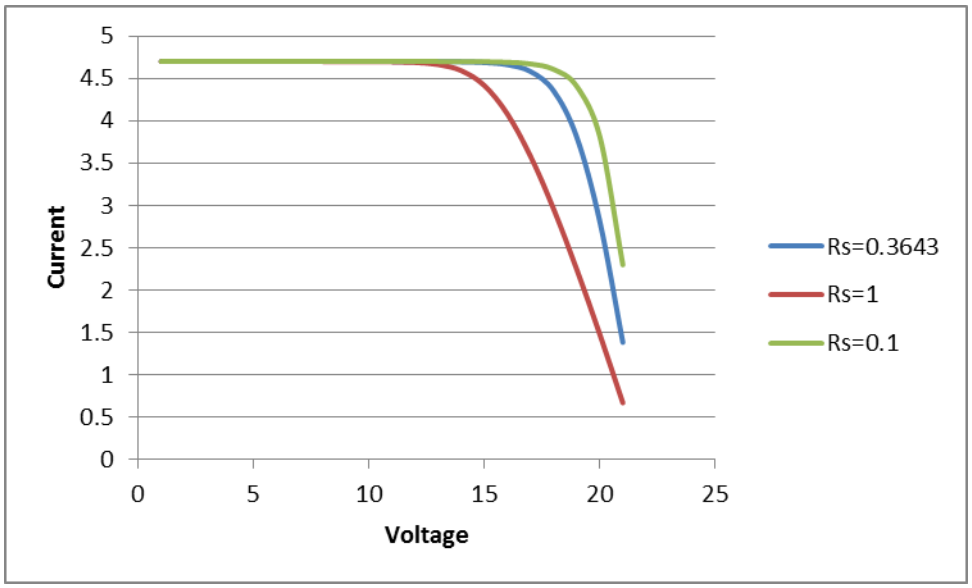


(a) Current-Voltage Curves

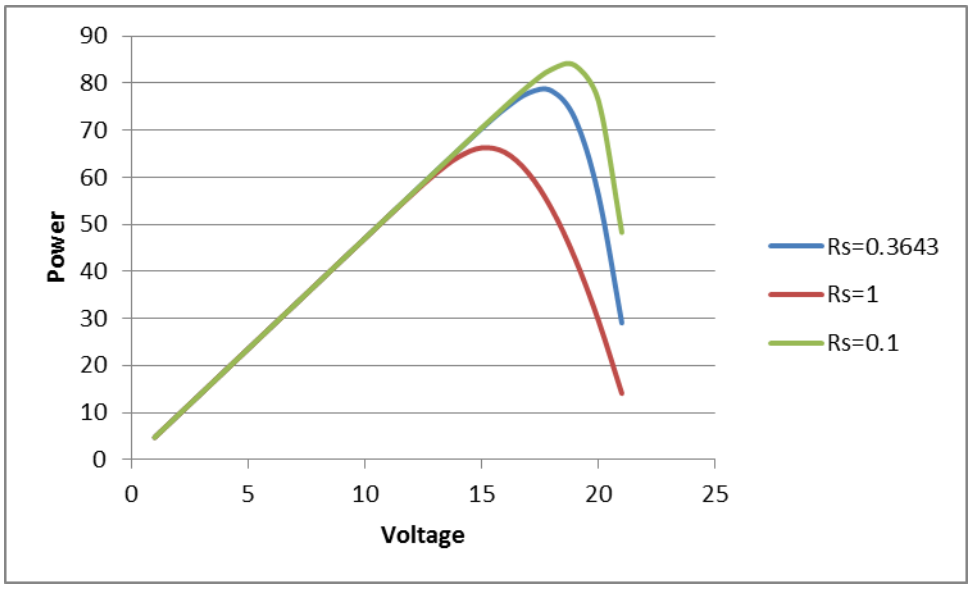


(b) Power-Voltage Curves

Figure 1.3 Effects of Shunt Resistance Variation



(a) Current-Voltage Curves



(b) Power-Voltage Curves

Figure 1.4 Effects of Series Resistance Variation

1.1.3 Maximum Power Point Tracking (MPPT). Numerous MPPT approaches have been implemented and discussed in the literature. It is difficult to over emphasize the importance of MPPT in a PV system. The basic concept of the MPPT is to operate at both I_{MPP} and V_{MPP} to obtain the most output power from a PV system at MPP. Esram and Chapman [10] summarized and classified various methods in 2007 referring to 90 journal/conference papers. Several others papers have been published since then. Choosing the most appropriate MPPT technique depends on the application, budget, and the user's knowledge. Some methods require a background in programming. Others require extensive knowledge in analog circuitry. For example, the performance and efficiency of an orbital space station are far more important than complexity and cost.

Tracking the MPP under a shading condition is complex; more than one MPP exists in the characteristic curves of the PV array. Tracking the real MPP when multiple local maxima exist is difficult with conventional methods. References [11-13] have discussed tracking an MPP in a PV array under the shading condition when several local maxima appear. These complex methods could be replaced by more efficient methods (discussed later in this thesis). One method suggests applying an individual MPPT across each PV panel to remove multiple local maxima problems. The other method is the parallel PV configuration with just one real MPP.

Perturbation and Observation is one of the most known commonly used MPPT methods [10, 14]. This method is preferred because it offers easy implementation. This method also has the ability of tracking the maximum power point of solar arrays under a large variation of atmospheric conditions such as array temperature or solar radiation. The P-V curve illustrates that, if either the voltage or the current is increasing, the output

power of the PV panel will increase as well until it reaches the MPP. After reaching the MPP, further increase in either the voltage or the current will decrease the output power. The concept here is to periodically add a perturbation (either an increase or a decrease) to the array terminal voltage (or current) and observe the output power in response to that perturbation. If the output power increases, a perturbation in the same direction will be added. If the output power decreases, a perturbation in the opposite direction will be added. This algorithm is repeated until the MPP is reached.

Connecting either the DC-DC converters or the micro-inverters across each PV panel would allow the array to operate in its MPP under shading conditions [15-17]. Petit and Aillerie [15], discussed the connection of DC-DC converters individually across each PV panel as illustrated in Figure 1.5. The purpose of this converter is to generate the optimum DC voltage of each PV panel, thereby adapting the rated voltage of the DC-AC inverter of the entire array. Petrone and has team [17], depicted the micro-inverter method as illustrated in Figure 1.6 that is, primarily, dedicated to a grid connected to a PV system. In this case, the micro-inverter is connected to each PV panel in the charge of boosting the low DC voltage of the panel to peak value of the AC grid.

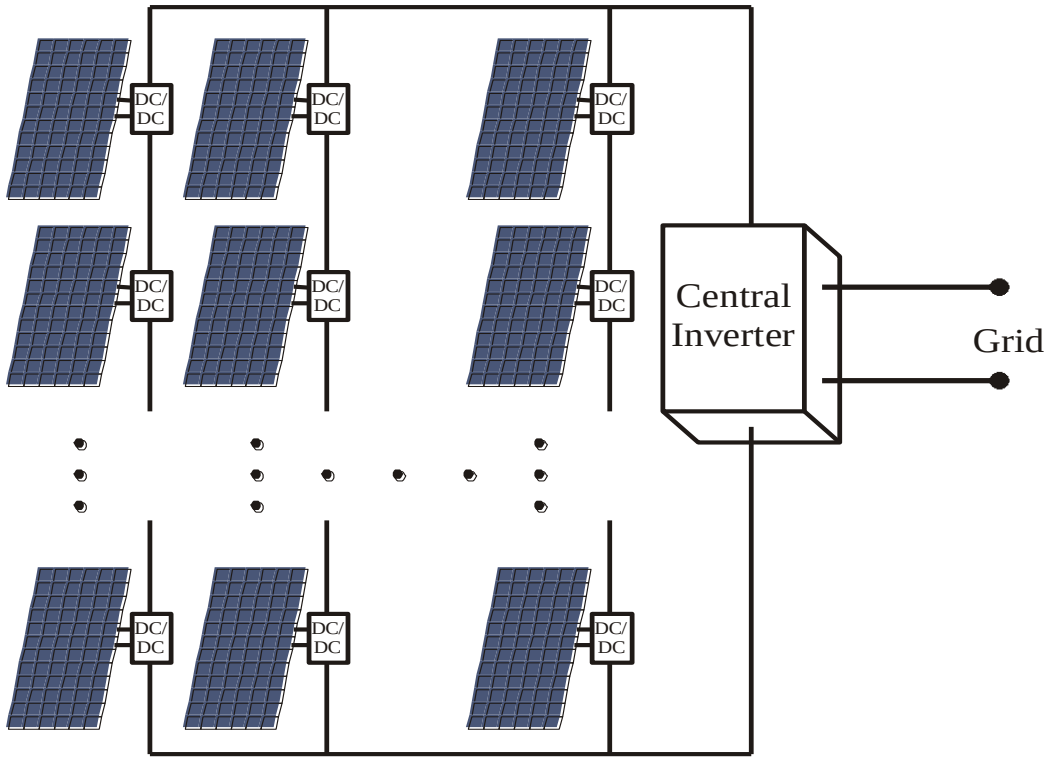


Figure 1.5 DC-DC Converters Across Each PV Panel [18]

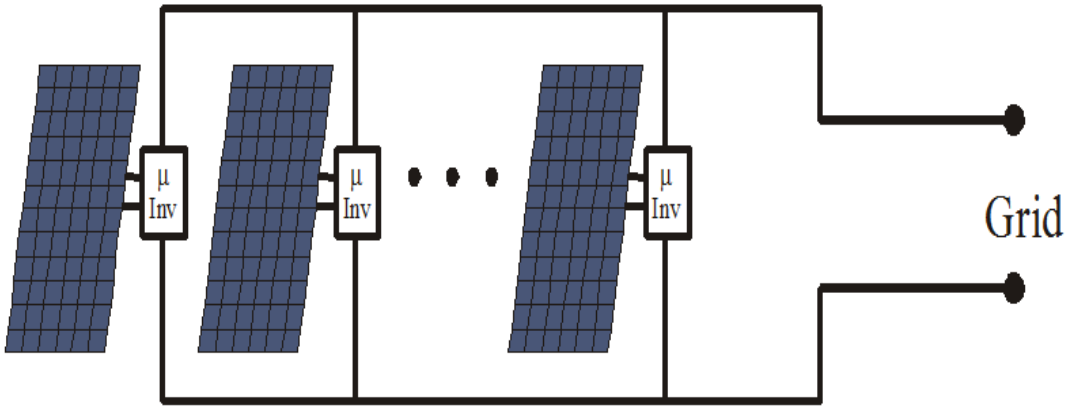


Figure 1.6 Micro-inverter Across Each PV Panel [18]

In this thesis a different method was suggested which allows the array to operate in its MPP under shading condition and improves the efficiency. The proposed method suggests connecting panels in parallel with DC-DC converters across each PV panel as illustrated in Figure 1.7. A comparison between this method and the conventional series configured PV panel is studied in this thesis. The comparison helps to find the advantage of this method to validate our assumption and identify the effectiveness of this method.

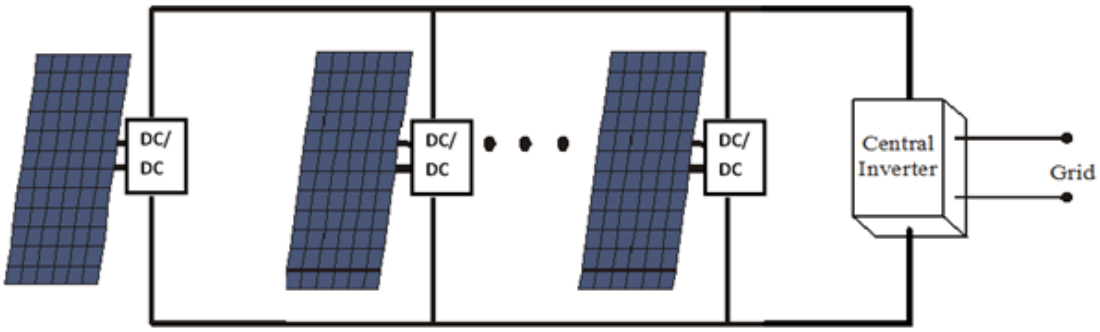


Figure 1.7 Parallel Connected with DC-DC Converters across Each PV Panel

2. EXPERIMENTAL METHODOLOGY

This section focuses on the structure of the sensor array used to capture solar radiation. This work was done by Faris Alfaris [19] and Beth Yount [20]. The sensor array used for this project was designed to be the same physical size as the 24 Kyocera KD135GX 125W panels setup in a traditional series parallel configuration. The array was constructed with both hinges and adjustable legs to collect the data at 0^{o} and 26^{o}, ending up being pitch of a typical household roof. The array was 17.5 ft. x 10 ft. 24 sensors were spread over the array and arranged in three rows: A, B, and C as shown in Figure 2.1

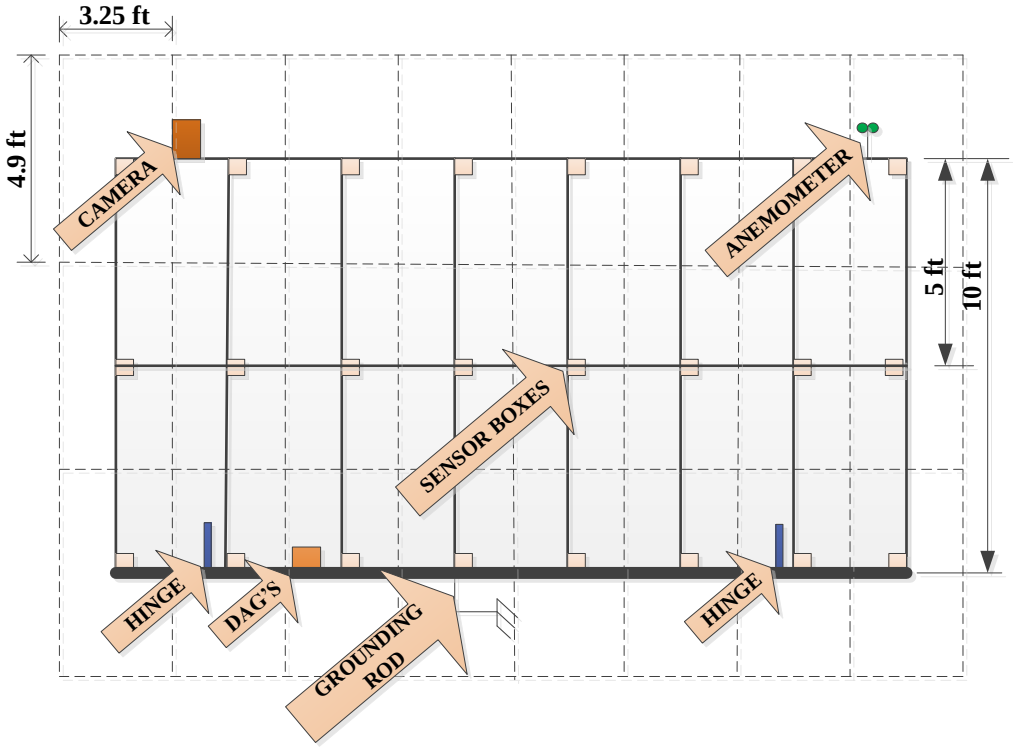


Figure 2.1 Solid Line Represents the Schematic Diagram of the Sensor Circuit Board and the Dash Line Represents the Actual Size of Kyocera Modules

Each sensor consisted of a solar cell, a circuit board, a 9V dry battery, and a CAT5e cable. All solar cells were typical and made by Solar Made [21] with 0.5V and 100mA rated voltage and current, respectively. Each solar cell was 2.5 cm. x 2 cm. The circuit board was designed to read four main quantities: short circuit current, open circuit voltage, load current, and weather temperature. These four values were transmitted to DAQs (Data Acquisition Systems) through a CAT5e cable to be recorded in a compact flash memory every other second. The schematic diagram of the circuit board is illustrated in Figure 2.2.

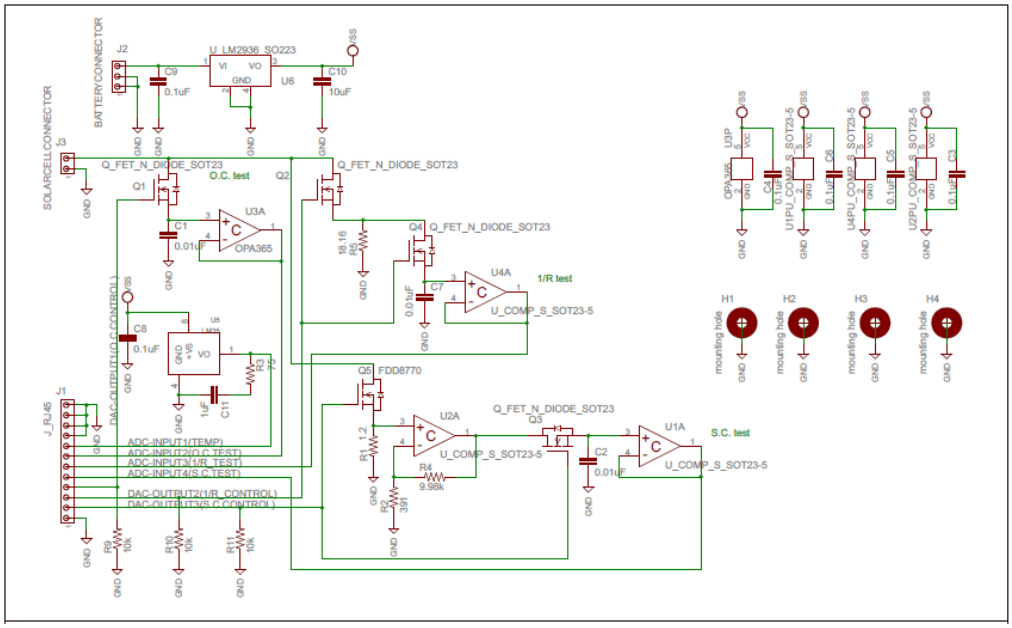


Figure 2.2 Schematic Diagram of the Sensor Circuit Board

All sensors were placed in a plastic box with a clear cover to allow maximum radiation. This box also protected sensors from the weather. The DAQs used were R-Engine-A programmable controller and produced by Tern Inc. [22]. The collected data

was saved in compact flash cards in Microsoft Excel spreadsheet format as a .csv extension. Stored files were analyzed in detail using MATLAB. Both current and voltage were transmitted (a short circuit current, a load current, open circuit voltage and temperature).

3. SOFTWARE DEVELOPMENT

Software was developed to process the data that is obtained from experiments. This software was developed using MATLAB with a GUI (graphical user interface). A GUI allows users to perform tasks interactively through controls, such as buttons and sliders. When used with MATLAB, a GUI allows the user to perform a number of operations on the plotted diagrams such as creating, designing, editing and enlarging the plots. This GUI also allows other operations such as curves, surfaces fitting and the analyzing and filtering the signals. As the phrase “user interface” indicates, the GUIs can be used within MATLAB or they can be run and operated separately outside the MATLAB [23]. However, the software only uses the GUI inside MATLAB.

The equations of photovoltaic (PV) currents, voltages and other parameters are cumbersome and time consuming when evaluated by ordinary numerical methods. The computation of these methods is made easy and less time consuming with the help of the designed software. This software is capable of taking sun radiation data as its input and produces an output that defines and describes the electrical characteristics curves of the entire panels.

Earlier work from Nisha Nagarajan [24] was analyzed and after careful considerations, the following solution is suggested that is thoroughly complete, has reduced time complexity and increased user-friendliness. The project includes 13 MATLAB scripts, each capable of calling the other. The function of each script is described below in detail:

1 Main Menu: The first file must be run to input some necessary information. This information should be defined before all of the files are run. User uses this menu to input the critical data that will be used in later modules. The common decisions made in this menu are:

a. Type of panel to simulate,

b. Number of panels are in series

c. Number of strings are in parallel

d. Use of a converter

e. The resistance and the gain of the converter incase if user chooses to work with the converter.(This kind of converter is a good model of the switched capacitor converter [25].)

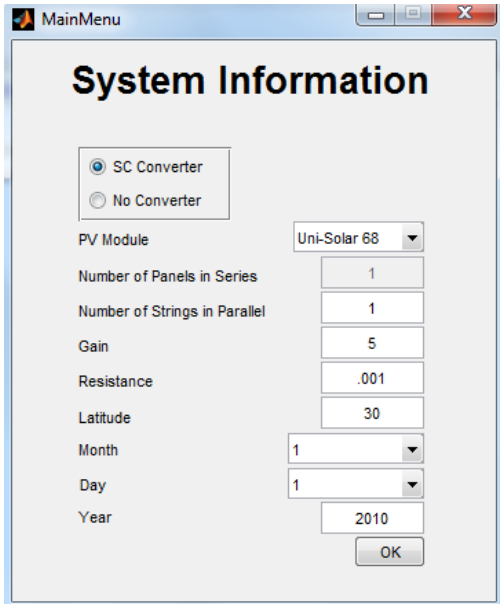


Figure 3.1 Main Menu

2. Insolation Menu: After the type and the configuration of the panel are chosen, the user is taken to insolation menu. The insolation menu provides following choices to input the sun radiations data: a) random, b) manual c) read from a file.

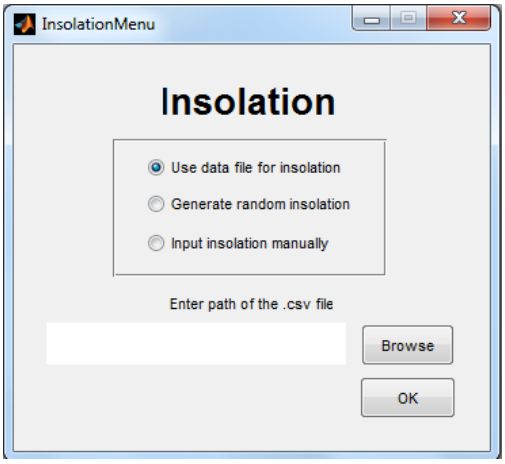


Figure 3.2 Insolation Menu

3. InputMatrix: The interface below will appear, if the user chooses to input the data manually.

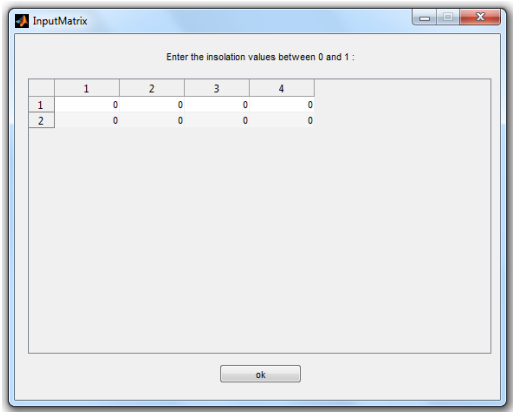


Figure 3.3 Editor for Entry

4. Random: If the user wants to run the software with random data, he/she needs to choose the maximum and minimum values of the insolation (scaled from 0 to 1, representing 0 to 1000 W/m^2). The software will generate random insolation in display.

5. Ifcustom: If the user does not choose one of the panels saved in the software, he/she must both define and input the required parameters he/she wants to simulate.

6. Irradiation: This file performs further operations, according to the choice taken in the 'Insolation Menu'. In case user chooses to access the data from Excel file, the Excel file must be saved with .csv (comma-separated values) extension.

7. Power Calculations: The main file is capable of taking all of the input data from the previous files. This file calls the rest of the files to makes the necessary calculations for the power and display the result.

8. Set Parameters: All of the input parameters that have been read before (e.g. resistance, gain, numPanels, sun radiation, Isc, Voc, numCells, A, Rs and Rsh) work as an input to this file. These parameters are then arranged in a Matrix form.

9. Series Calculation**:** The series calculation file calculates both the current and the voltage for series-connected PV panels.

10. Newton PV: The Newton PV file will be called by a Series Calculation to solve the equation using Newton's method.

11. Current Calculation: The current calculation solves the PV equations using the '*fzero'* function.

12. Auxiliary: The auxiliary file is used to arrange the equations within a current's function to solve it using a Current Calculation.

13. Differential Calculation: A differential calculation file is used to input values to the Jacobian matrix necessary for calculations.

3.1. SOFTWARE THEORY OF OPERATION

3.1.1. Input. 'Main Menu' is the first file that needs to be run. The outputs from the 'Main Menu' are the type of the panel, the number of the panel in a series, and the strings in parallel. At the end of the 'Main Menu', the 'Insolation Menu' file will be called. The function of 'Insolation Menu' is to choose from where the data needs to be read (Excel file, random generated, manually entered). The 'Insolation Menu' file will call the main file, called 'Power Calculations'. The software will take all of the input data and call the Irradiation file. The software will then either execute function 'Rand (Max, Min)' or Read the data from Excel file, or read the data from the input matrix.

3.1.2 Analysis. Suppose the software is still running in 'Power Calculations'. 'Power Calculations' read the data from the previous files and then do the analyses. The term 'ti' is the number of samples that are read from the excel file. Analysis for every sample is accomplished before the maximum value is taken. The current-voltage calculation is done for every 'ti'. A separate file called 'Set Parameter' is also called to arrange all of the parameters in to a matrix form. Once complete, the 'Series Calculation' will be called to calculate both the current and the voltage for all of the panels within the

series. Both the output current and the voltage (for the entire panels) will be calculated after the series connection is calculated.

3.1.3 Case Study. For the clarification of the description above, case study is now presented below. When the software is run, Main Menu appears as shown in Figure 3.4. The user enters parameters in to the input boxes before pressing the OK button. The type of panel can also be selected by the user at this menu.

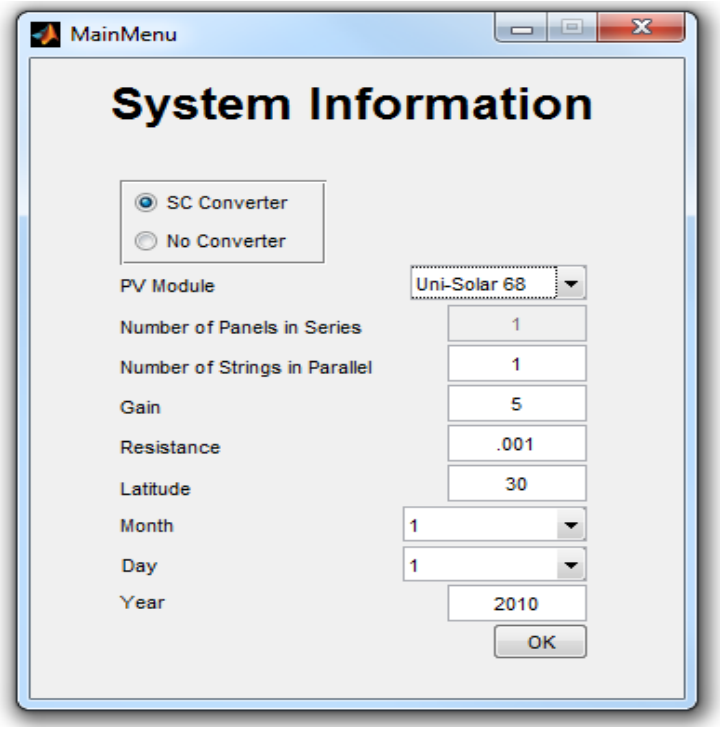


Figure 3.4 Main Menu

This software can offer five standard photovoltaic modules, as shown in Figure 3.5. The selection button is offers the user five pre-defined panels. The last selection is a

custom module representing an undefined panel. If the user prefers to choose custom module, Parameter entry interface appears as shown in Figure 3.6. The parameters for defined panels are given in Table3.1.

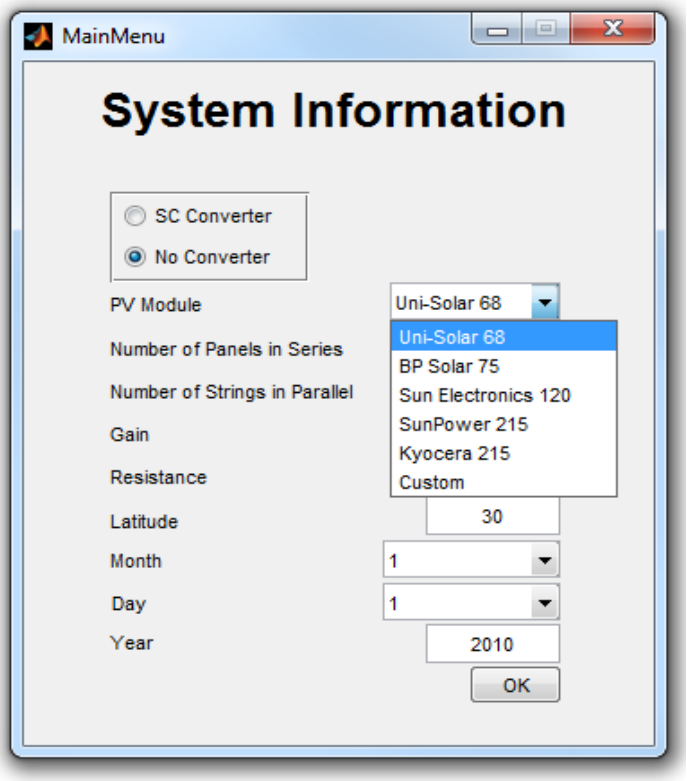


Figure 3.5 PV Module Selection Pop-up Menu

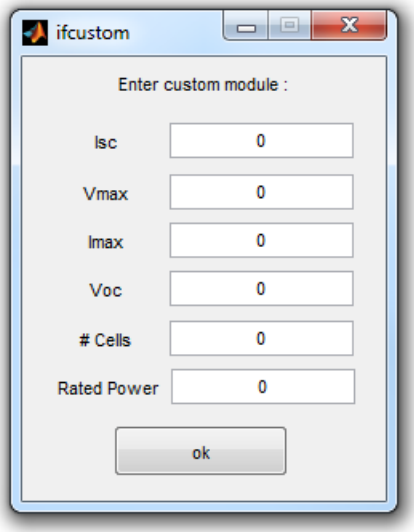


Figure 3.6 Parameter Entry Interface

Table 3.1 The Panels Saved in the Software

	I_{SC}	V_{mpp}	I_{mpp}	V_{oc}	# No of Cells	Rated Power
Uni-Solar 68	5.1	16.5	4.1	23.1	11	68
BP Solar 75	4.7	17.3	4.3	21.8	36	75
Sun Electronics 120	7.62	17.27	6.95	21.34	45	120
SunPower 215	5.8	39.8	5.4	48.3	72	215
Kyocera 215	8.78	26.6	8.09	33.2	54	215
Custom	User can define parameters of any panel					

The Uni-Solar 68 panel was selected as the sample for this case study to perform the simulation process without making use of any inverter. At the beginning, both the configuration and panel type are selected by the user. Next, Insolation menu appears as shown in Figure 3.7. Where the following options are available: 1) take data from the file, 2) generate random insolation using a MATLAB built-in function, and 3) define manually.

In the first simulation, Insolation data was generated randomly; the user had to enter data in the range of 0 to 1 which represent 0 to 1000 W/m^2 solar radiation. In the second simulation data was entered manually.

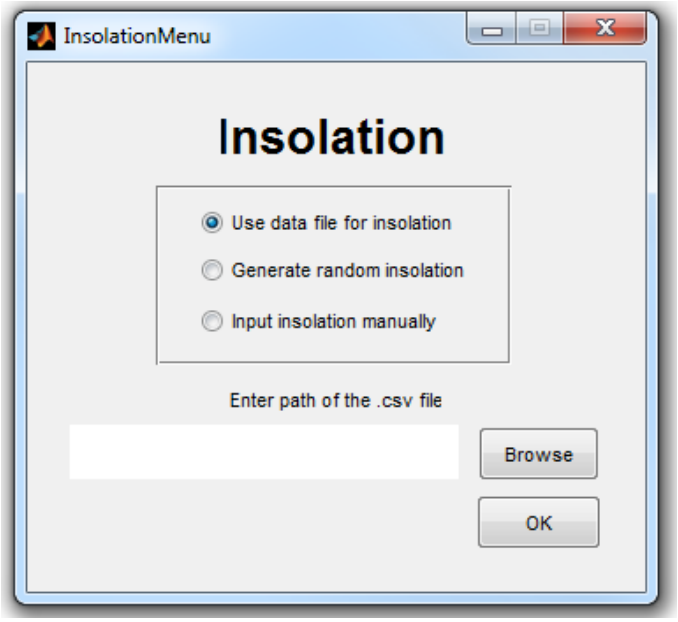


Figure 3.7 Insolation Menu

In Figure 3.8, the user asked the software to generate insolation randomly. In Figure 3.9, the user entered the insolation data manually.

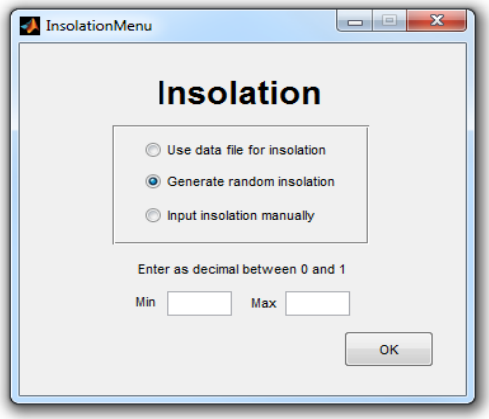


Figure 3.8 Random Generation

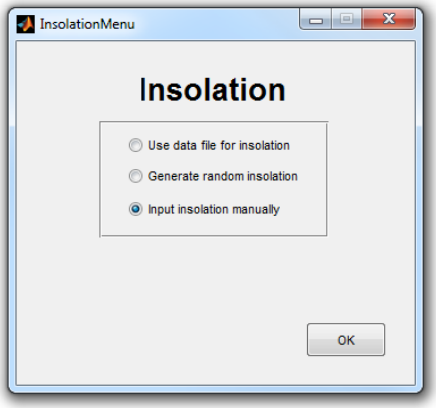


Figure 3.9 Manual Insolation Input

Random Insolation generated was between 800W/m^2 and 1000W/m^2, as defined by the user. The value selected by the software was 855.6996W (see Figure 3.10). The editor can also enter the sun radiations value manually (see Figure 3.11).

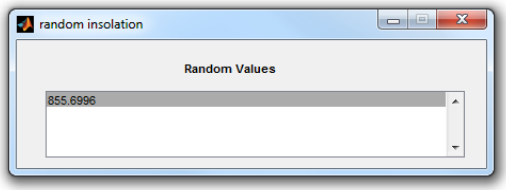


Figure 3.10 Random Sun Radiation Value

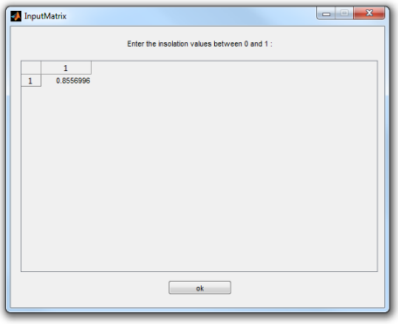


Figure 3.11 Editor for Entry

After having the PV configuration and the data, the software will present user with the output as in Figures 3.12-3.15.

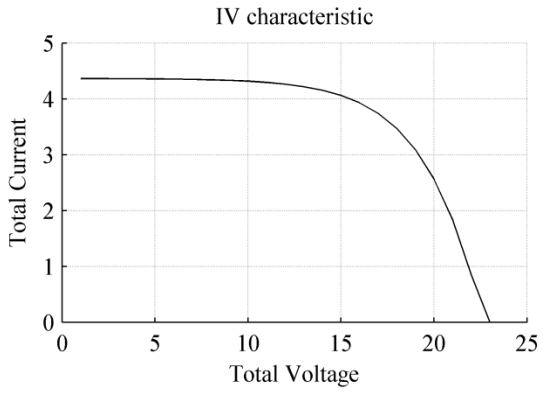


Figure 3.12 I-V Characteristic Curves

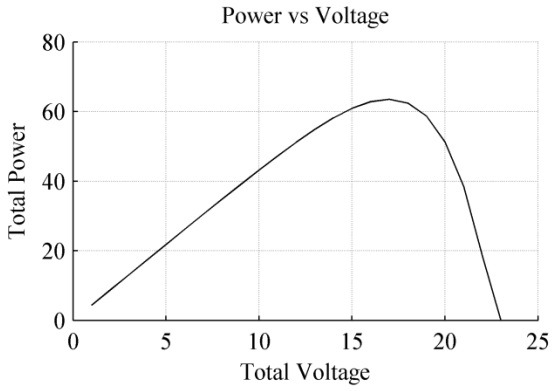


Figure 3.13 Power versus Voltage Curve

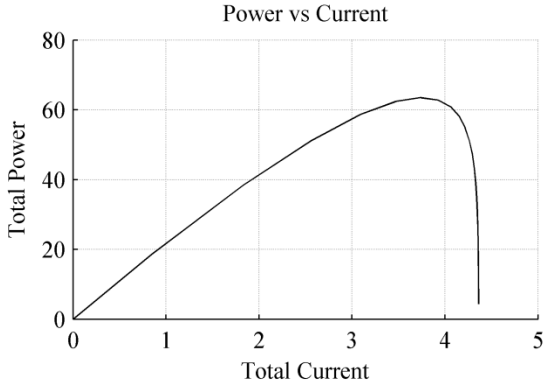


Figure 3.14 Power versus Current Curve

4. CALCULATED PERFORMANCE

The goal of this work was to determine the best possible solar panel configuration. Various scenarios were studied to accomplish this task. Each scenario was based on a different climatic condition. Outputs from each scenario were formulated in tables. These tables illustrate the percentage difference of the output energy, in various configurations, of a solar array (series-parallel and parallel-connection with an IPC). This difference was calculated using

$$\%Difference = \frac{E_s - E_p}{E_s}$$

where E_s is the output energy for 2S4P, 4S2P, and 8S1P. E_p is the output energy of 1S8P connected to an IPC. "2S4P" is 2 series panels per string on 4 parallel strings.

4.1. SCENARIO 1

The data taken from the sensor array on June 3, 2012 is analyzed in Scenario 1. Figure 4.1 illustrates the radiation measured from the sensors used in the analysis. Panel Uni-Solar 68 was used in the simulation. A total of 532 samples from the sensors were used. Each sample was picked in every 2 minutes.

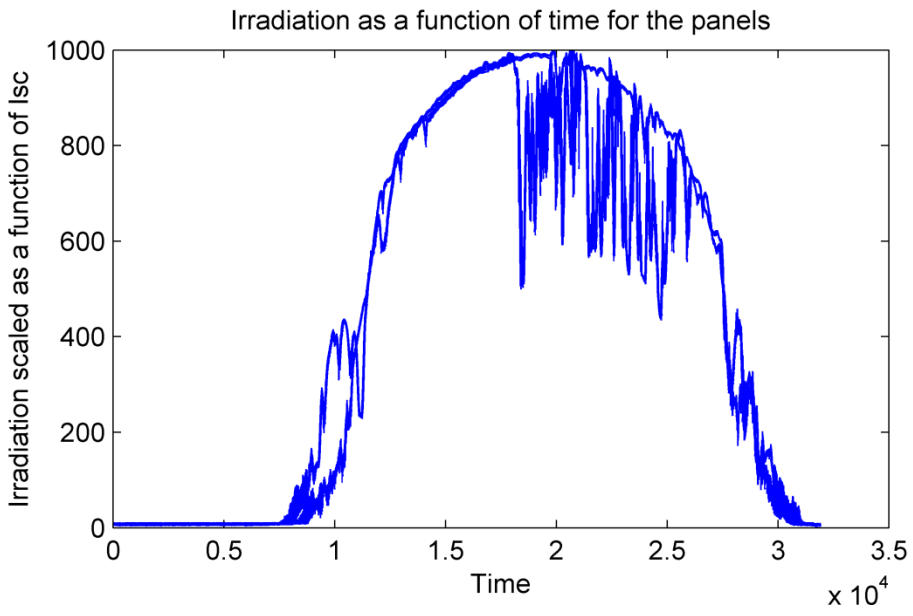


Figure 4.1 The Insolation Data as a Function of time

Case 1: The first case in Scenario 1 compared the 2S4P configuration without an IPC connection to 1S8P configuration with each panel connected to an IPC. The difference in output energy can be seen in Table 4.1. The output energy for 1S8P with an IPC is 3.436% (with the 0.8736Ω resistor) more than 2S4P without an IPC. The characteristic curves of PV panels 2S4P and 1S8P are given in Figure 4.2 and 4.3 respectively.

Table 4.1 The Difference in the Output Energy (2S4P and 1S8P)

	Number of Panels in Series	Number of Strings in parallel	Gain	Resistance	The Energy	% difference
No IPC	2	4	N/A	N/A	9064200	N/A
With IPC	N/A	8	10	40	8894640	-1.870
				30	9008880	-0.610
				16	9183840	1.319
				8.736	9276000	2.336
				4	9336240	3.001
				2	9361560	3.280
				0.8736	9375720	3.436
				0.6	9379080	3.473
				0.5	9380400	3.488
				0.25	9383520	3.522
				0.1	9385440	3.544

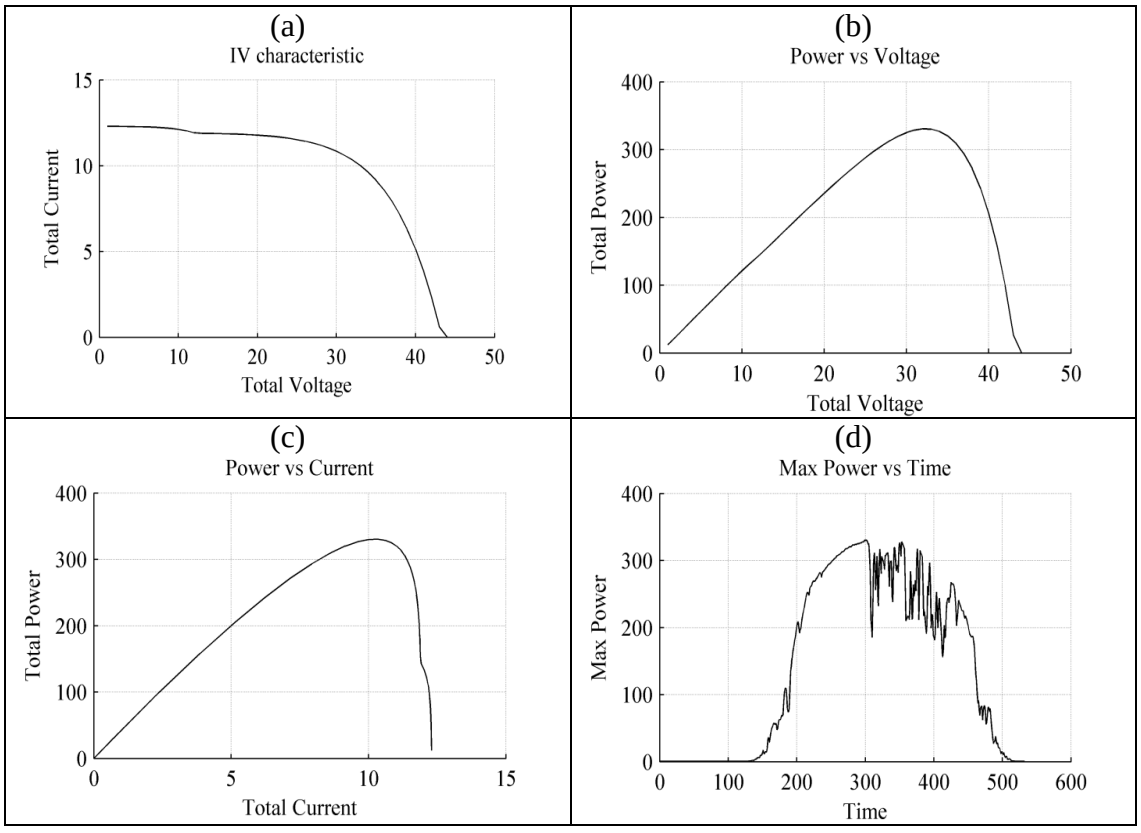


Figure 4.2 Array Characteristics of (2S4P) Without IPC

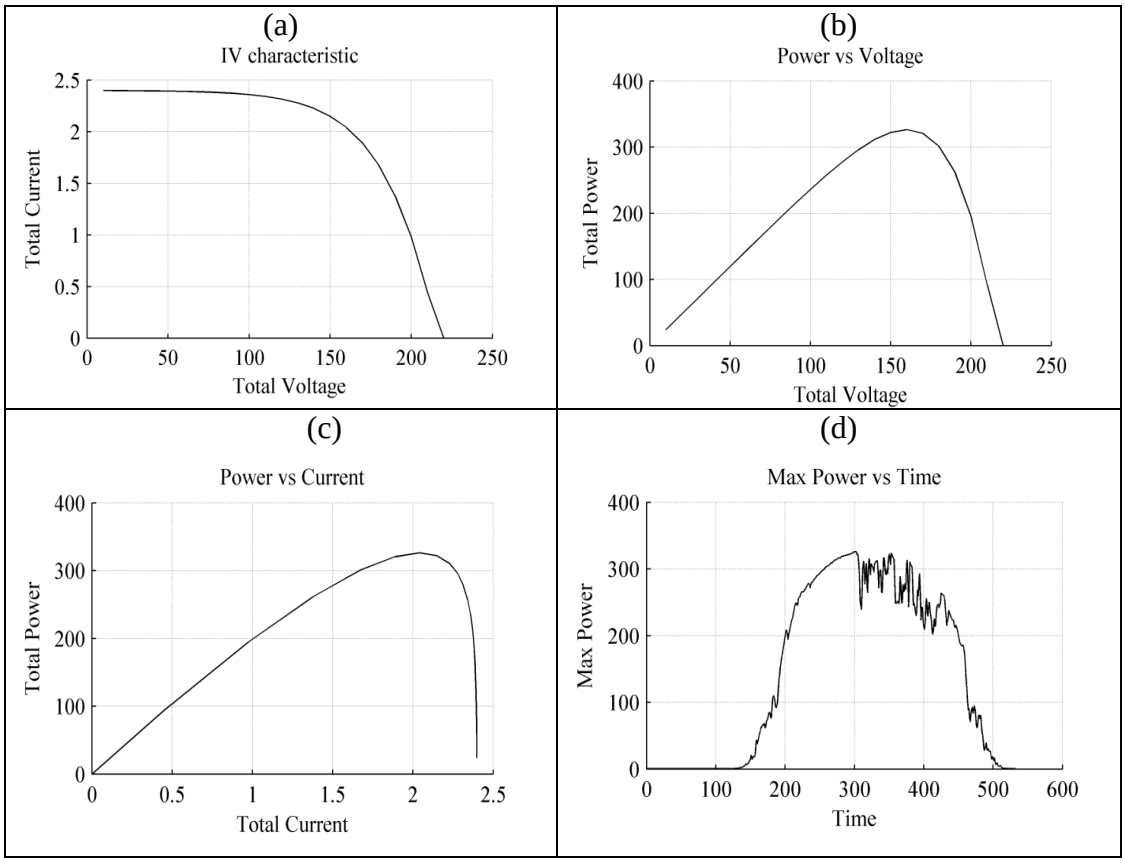


Figure 4.3 Array Characteristics of (1S8P)

Case 2: The second case in Scenario 1 shows the difference between the configuration of a 4S2P without an IPC and a 1S8P with an IPC. The output energy of a 4S2P is less than 2S4P. So, there is more percentage difference in energy between 4S2P and 1S8P compared to 2S4P and 1S8P as shown in Table 4.2. In this scenario 4S2P and 1S8P will result 1.5~3.7% energy difference while 2S4P and 1S8P will result 1.3~3.5% energy difference. The characteristic curves of the PV panel 2S4P can be seen in Figure 4.4.

Table 4.2 The Difference in the Output Energy (4S2P and 1S8P)

	Number of Panels in Series	Number of Strings in parallel	Gain	Resistance	The Energy	% difference
No IPC	4	2	N/A	N/A	9043440	N/A
With IPC	N/A	8	10	40	8894640	-1.645
				30	9008880	-0.382
				16	9183840	1.552
				8.736	9276000	2.571
				4	9336240	3.237
				2	9361560	3.517
				0.8736	9375720	3.674
				0.6	9379080	3.711
				0.5	9380400	3.726
				0.25	9383520	3.760
				0.1	9385440	3.781

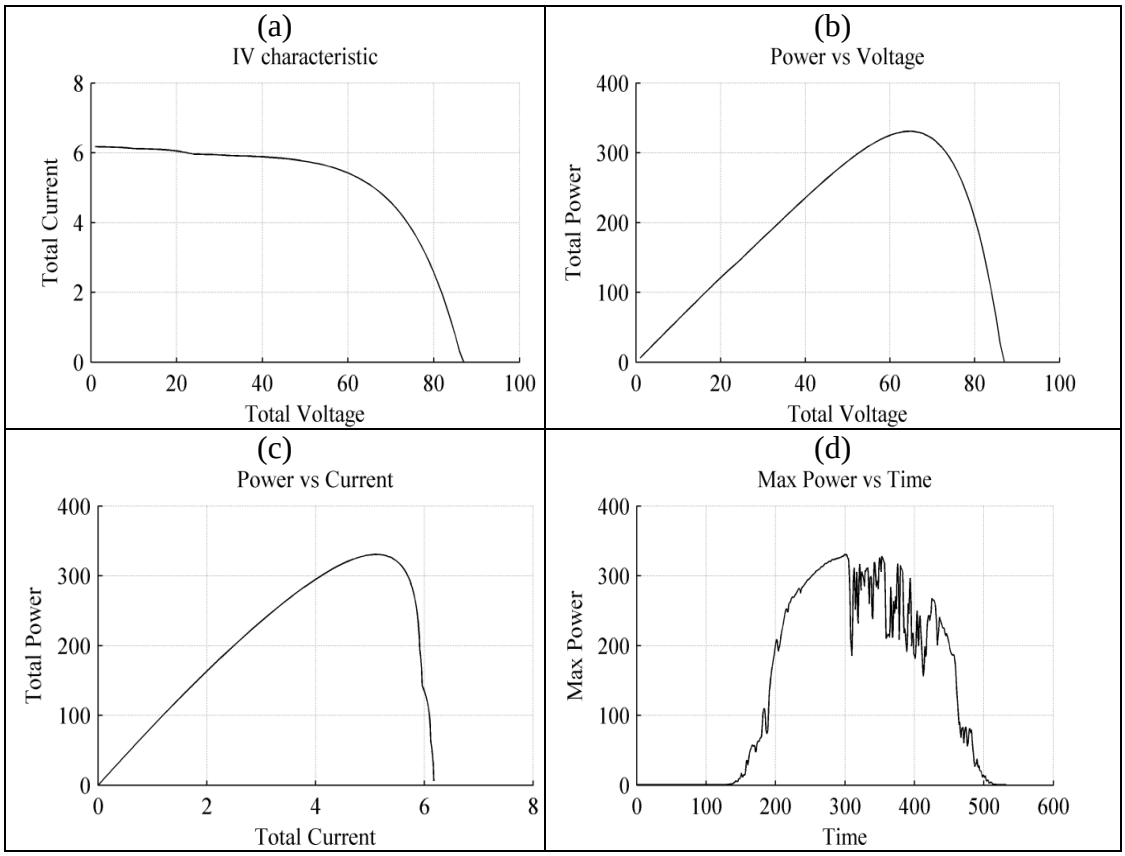


Figure 4.4 Array Characteristics of (4S2P)

Case 3: The third case in Scenario 1 illustrates the difference between the configuration of an 8S1P without an IPC and a 1S8P with an IPC. The output energy of a 1S8P is less than that of a 4S2P. So, there is more percentage difference in energy between 8S1P and 1S8P compared to 4S2P and 1S8P as shown in Table 4.3. In this scenario 4S2P and 1S8P will result 1.3~3.5% energy difference while 8S1P and 1S8P will result 1.684~3.916% energy difference. The characteristic curves of the PV panel 2S4P can be seen in Figure 4.5.

Table 4.3 The Difference in the Output Energy (8S1P and 1S8P)

	Number of Panels in Series	**Number of Strings in parallel**	**Gain**	**Resistance**	**The Energy**	**% difference**
No IPC	8	1	N/A	N/A	9031680	N/A
With IPC	N/A	8	10	40	8894640	-1.517
				30	9008880	-0.252
				16	9183840	1.684
				8.736	9276000	2.705
				4	9336240	3.372
				2	9361560	3.652
				0.8736	9375720	3.809
				0.6	9379080	3.846
				0.5	9380400	3.861
				0.25	9383520	3.895
				0.1	9385440	3.916

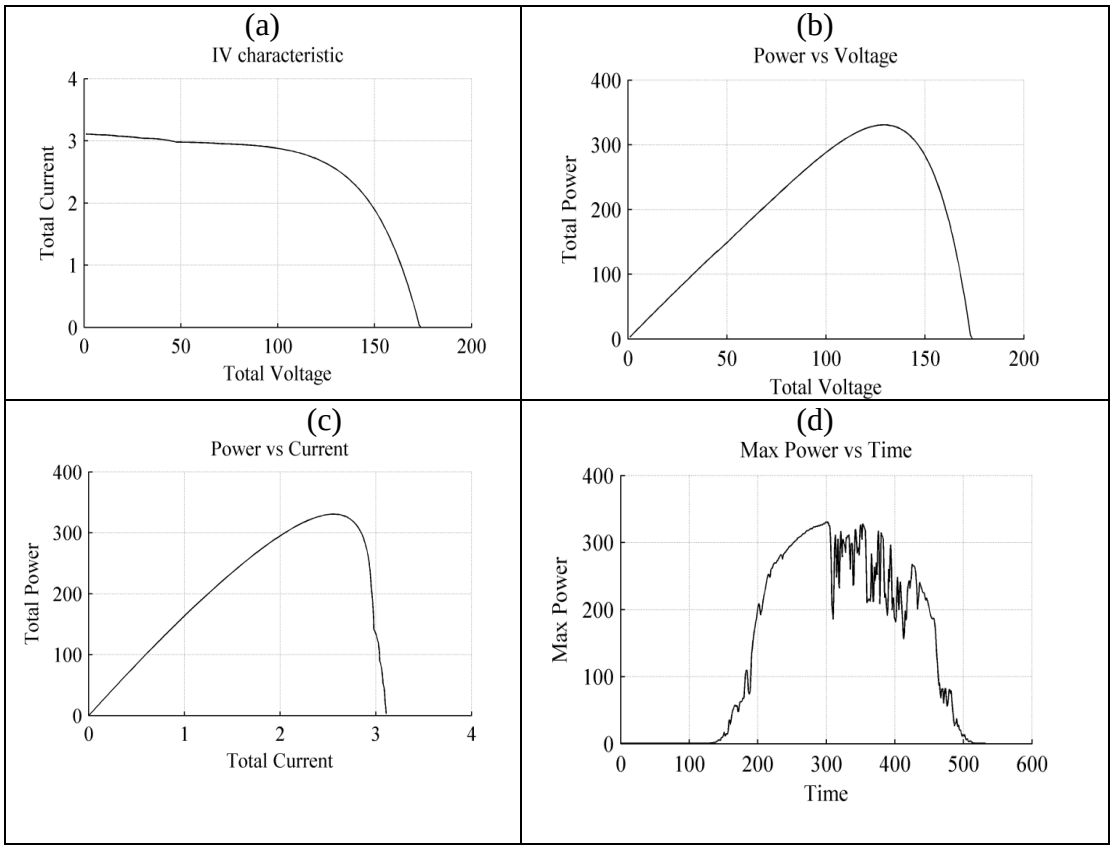


Figure 4.5 Array Characteristics of (8S1P)

The theoretical energy from eight individual panels (if individually connected to MPPTs) was calculated to be 9,403,764 J. Figure 4.6 represents the output energy of the entire configurations: 2S4P, 4S2P, 8S1P and 1S8P divided by the theoretical value. The output energy of a 1S8P is seen decreasing as the resistance of the IPC increases.

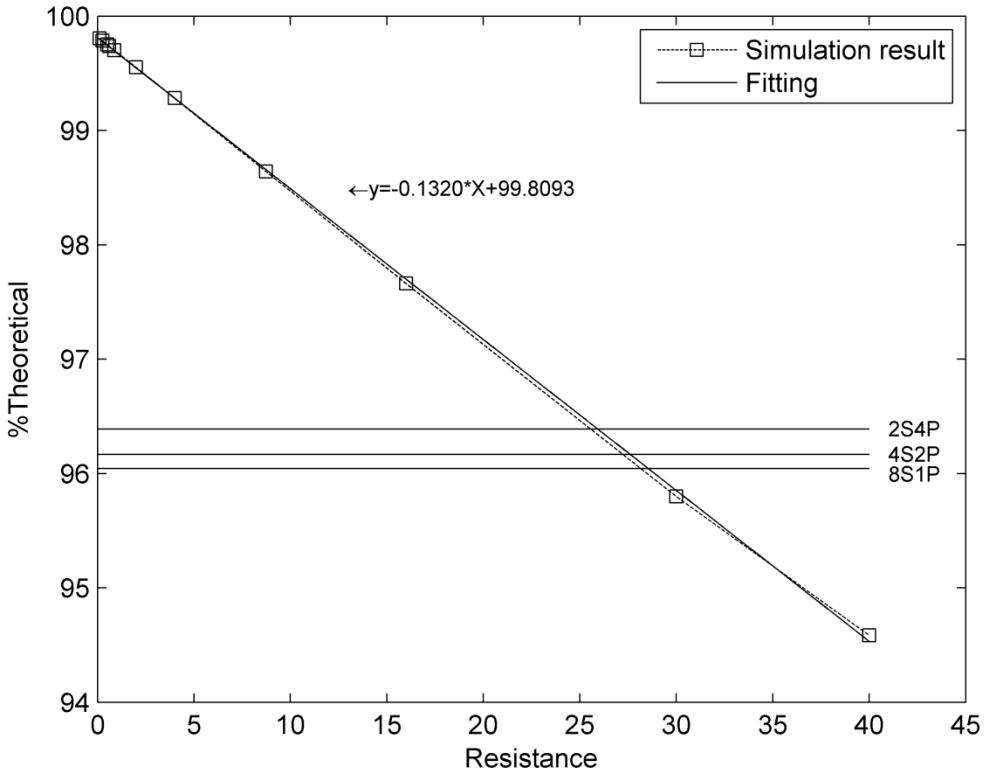


Figure 4.6 %Theoretical vs. Resistance

4.2. SCENARIO 2

The data taken from the sensor array on June 15, 2012 is analyzed in Scenario 2. Figure 4.7 illustrates the radiation measured from the sensors used in the analysis. Panel Uni-Solar 68 was used in the simulation. A total of 532 samples from the sensors were used. Each sample was picked in every 2 minutes.

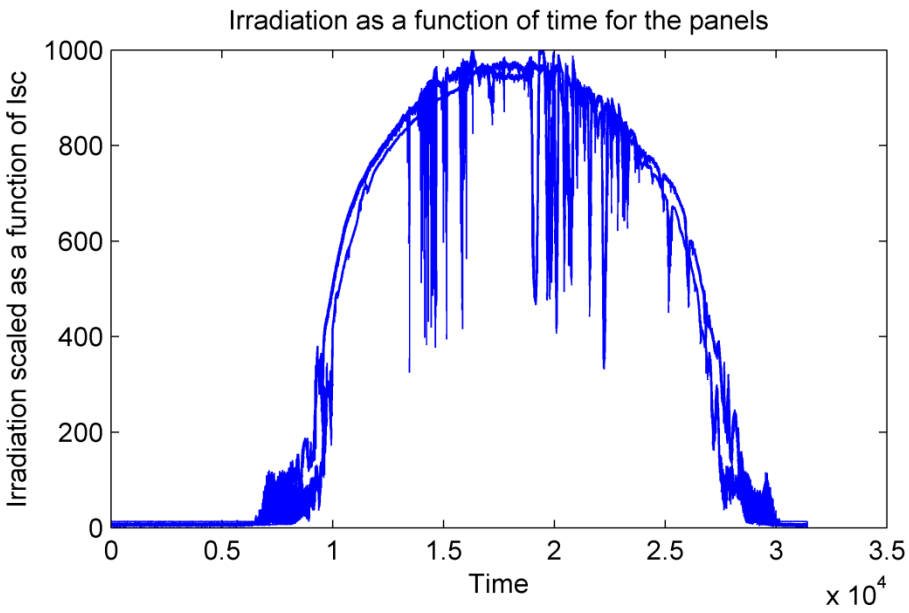


Figure 4.7 The Insolation Data as a Function of Time

Case 1**:** The first case in Scenario 2 compared the 2S4P configuration without an IPC connection to 1S8P configuration with each panel connected to an IPC. The difference in output energy can be seen in Table 4.4. The output energy for 1S8P with an IPC is 1.264% (with the 0.8736Ω resistor) more than 2S4P without an IPC. The characteristic curves of PV panels 2S4P and 1S8P are given in Figure 4.8 and 4.9 respectively.

Table 4.4 The Difference in the Output Energy (2S4P and 1S8P)

	Number of Panels in Series	Number of Strings in parallel	Gain	Resistance	The Energy	% difference
No IPC	2	4	N/A	N/A	9422640	N/A
With IPC	N/A	8	10	40	9034320	-4.121
				30	9155760	-2.832
				16	9340320	-0.873
				8.736	9437640	0.159
				4	9500760	0.829
				2	9527040	1.107
				0.8736	9541800	1.264
				0.6	9545400	1.302
				0.5	9546720	1.316
				0.25	9549960	1.351
				0.1	9552000	1.372

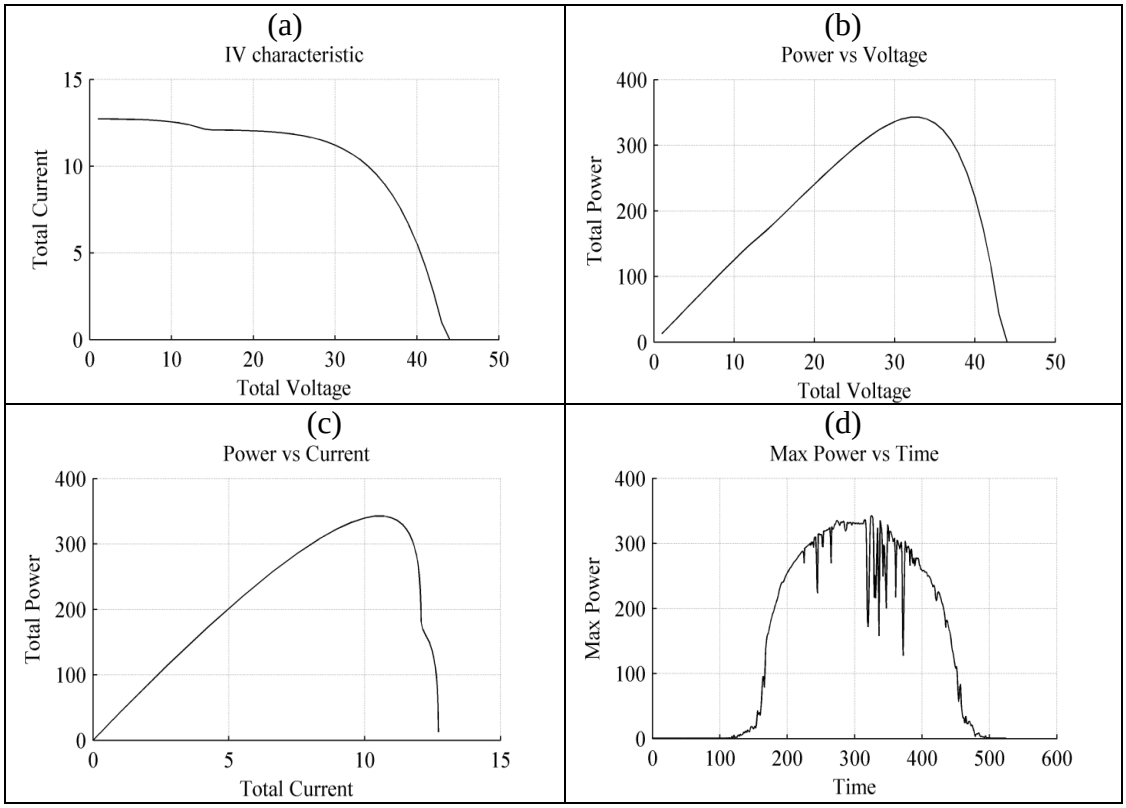


Figure 4.8 Array Characteristics of (2S4P) Without IPC

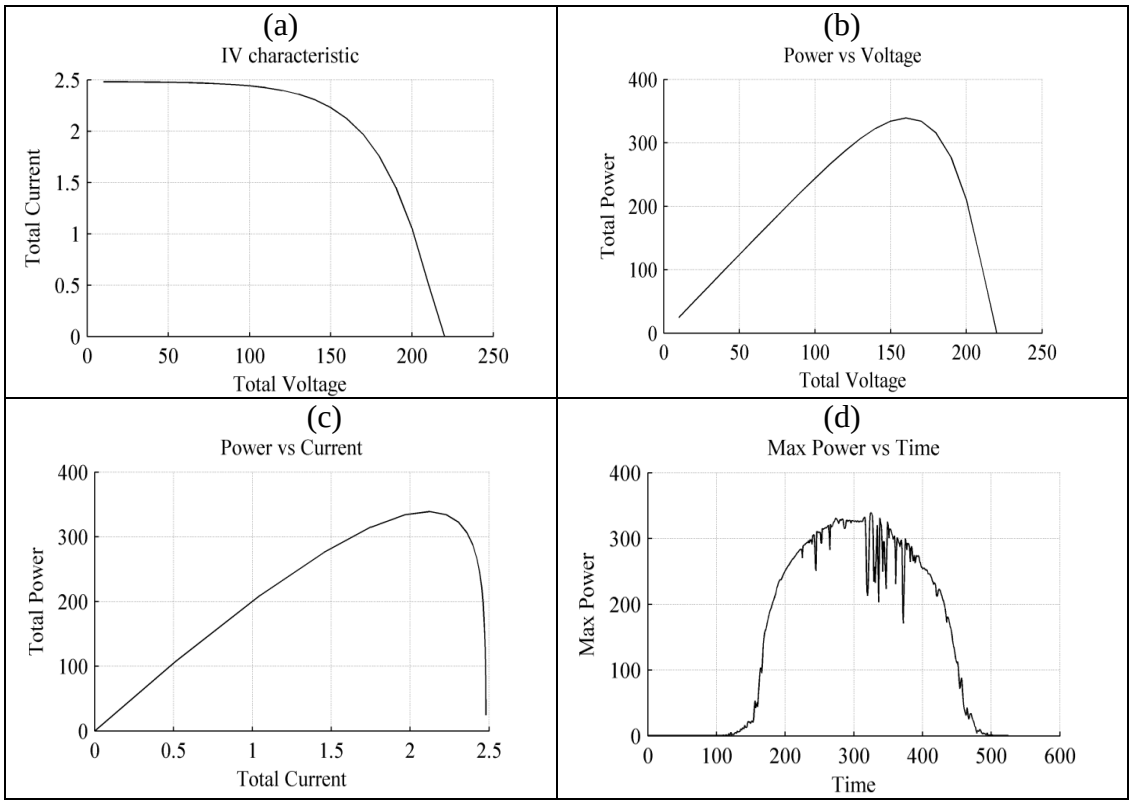


Figure 4.9 Array Characteristics of (1S8P)

Case 2: The second case in Scenario 2 shows the difference between the configuration of a 4S2P without an IPC and a 1S8P with an IPC. The output energy of a 4S2P is less than 2S4P. So, there is more percentage difference in energy between 4S2P and 1S8P compared to 2S4P and 1S8P as shown in Table 4.5. In this scenario 4S2P and 1S8P will result 0.617~1.837% energy difference while 2S4P and 1S8P will result 0.159~1.372% energy difference. The characteristic curves of the PV panel 2S4P can be seen in Figure 4.10.

Table 4.5 The Difference in the Output Energy (4S2P and 1S8P)

	Number of Panels in Series	Number of Strings in parallel	Gain	Resistance	The Energy	% difference
No IPC	4	2	N/A	N/A	9379680	N/A
With IPC	N/A	8	10	40	9034320	-3.682
				30	9155760	-2.387
				16	9340320	-0.419
				8.736	9437640	0.617
				4	9500760	1.290
				2	9527040	1.571
				0.8736	9541800	1.728
				0.6	9545400	1.766
				0.5	9546720	1.780
				0.25	9549960	1.815
				0.1	9552000	1.837

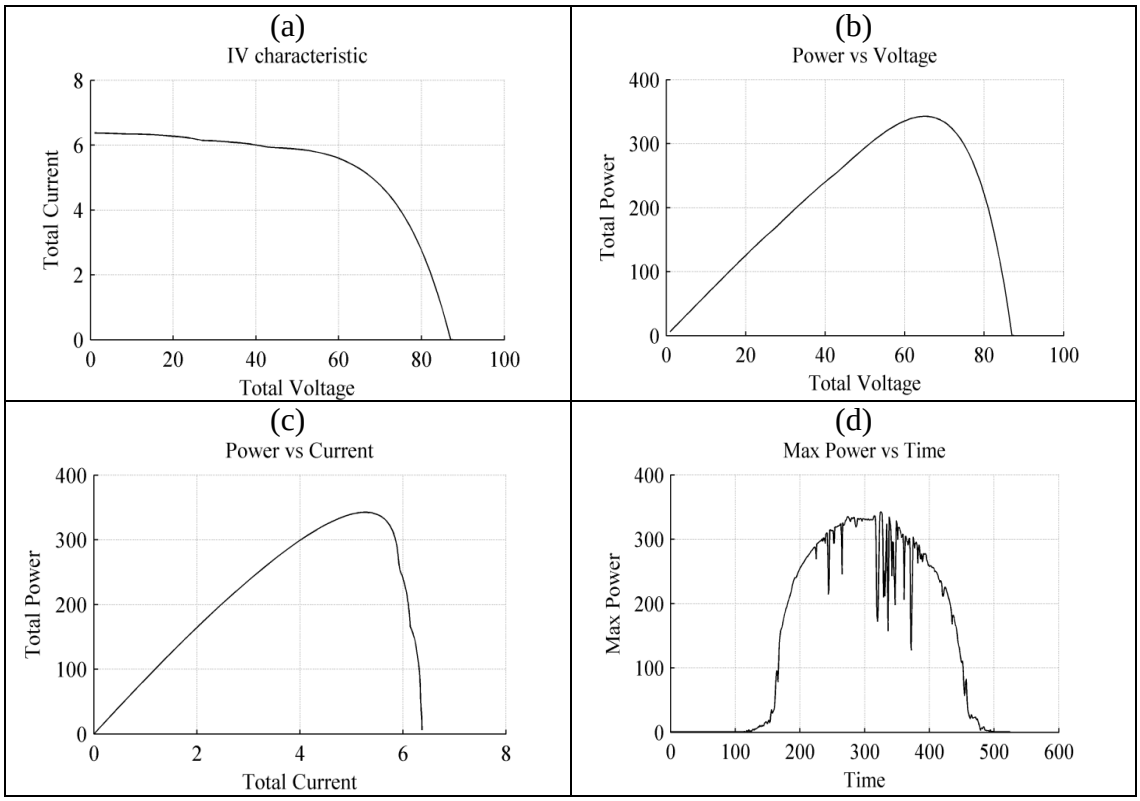


Figure 4.10 Array Characteristics of (4S2P)

Case 3: The third case in Scenario 2 illustrates the difference between the configuration of an 8S1P without an IPC and a 1S8P with an IPC. The output energy of a 1S8P is less than that of a 4S2P. So, there is more percentage difference in energy between 8S1P and 1S8P compared to 4S2P and 1S8P as shown in Table 4.6. In this scenario 4S2P and 1S8P will result 0.159~1.372% energy difference while 8S1P and 1S8P will result 0.843~2.065% energy difference. The characteristic curves of the PV panel 2S4P can be seen in Figure 4.11.

Table 4.6 The Difference in the Output Energy (8S1P and 1S8P)

	Number of Panels in Series	**Number of Strings in parallel**	**Gain**	**Resistance**	**The Energy**	**% difference**
No IPC	8	1	N/A	N/A	9358680	N/A
With IPC	N/A	8	10	40	9034320	-3.465
				30	9155760	-2.168
				16	9340320	-0.196
				8.736	9437640	0.843
				4	9500760	1.518
				2	9527040	1.798
				0.8736	9541800	1.956
				0.6	9545400	1.995
				0.5	9546720	2.009
				0.25	9549960	2.043
				0.1	9552000	2.065

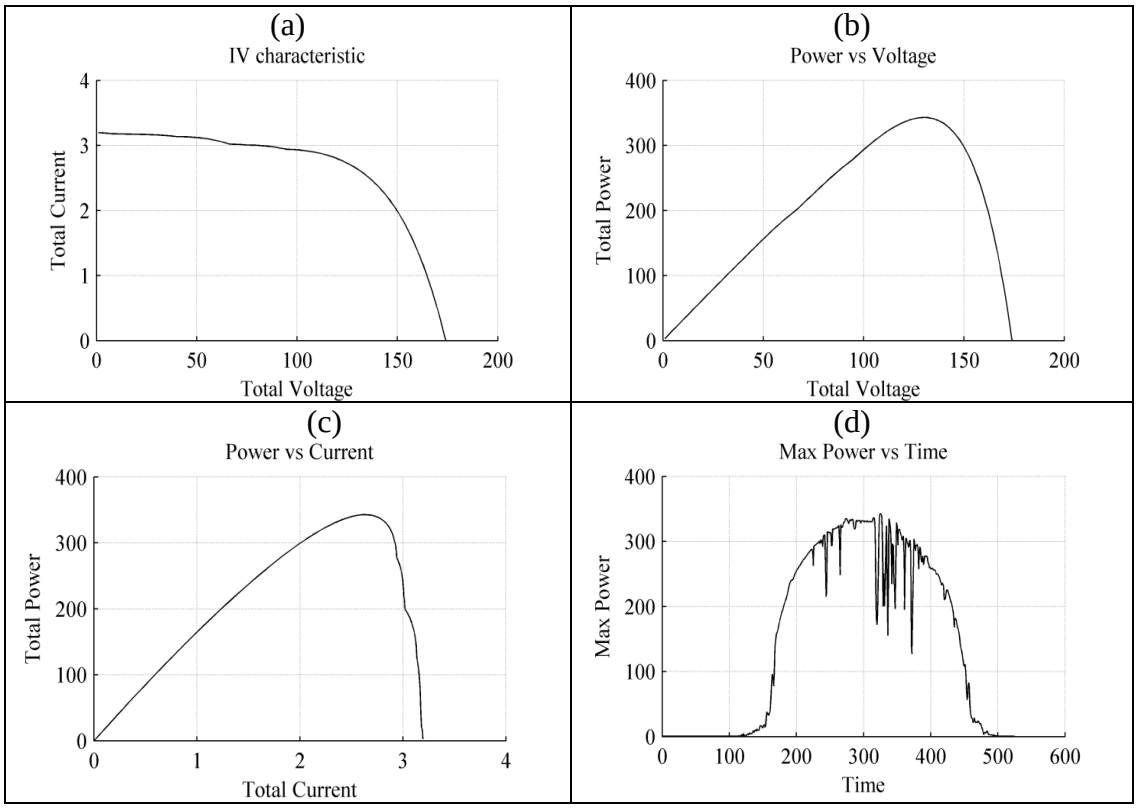


Figure 4.11 Array Characteristics of (8S1P)

The theoretical energy from eight individual panels (if individually connected to MPPTs) was calculated to be 9,564,108 J. Figure 4.12 represents the output energy of the entire configurations: 2S4P, 4S2P, 8S1P and 1S8P divided by the theoretical value. The output energy of a 1S8P is seen decreasing as the resistance of the IPC increases.

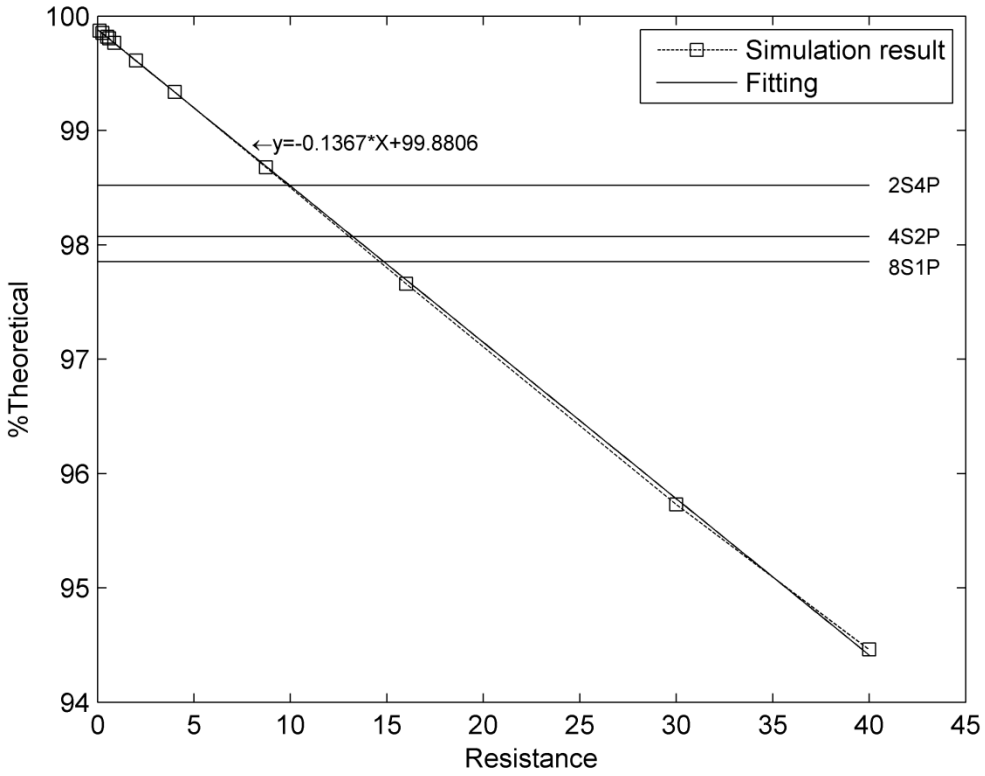


Figure 4.12 %Theoretical vs. Resistance

4.3. SCENARIO 3

The data taken from the sensor array on September 16, 2012 is analyzed in Scenario 3. Figure 4.13 illustrates the radiation measured from the sensors used in the analysis. Panel Uni-Solar 68 was used in the simulation. A total of 532 samples from the sensors were used. Each sample was picked in every 2 minutes.

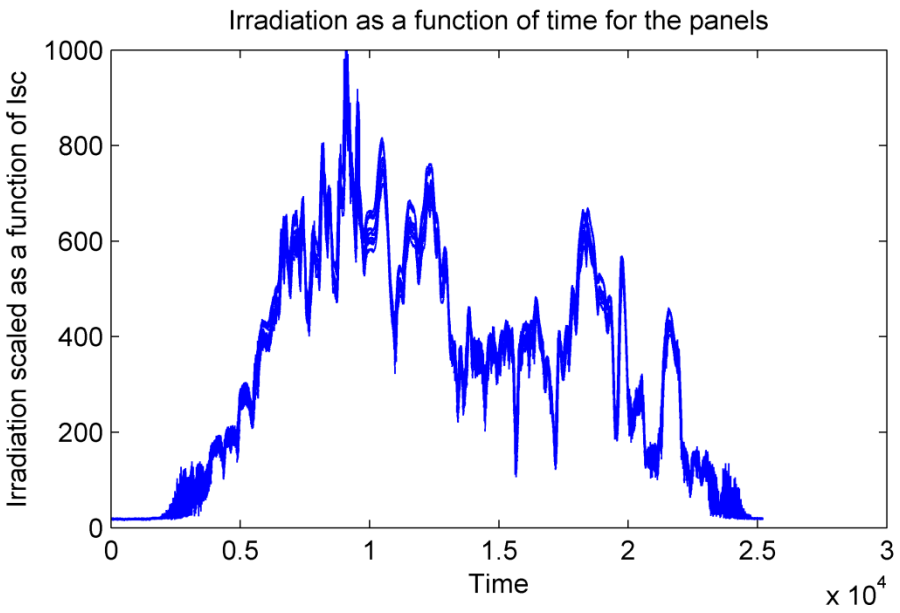


Figure 4.13 The Insolation Data as a Function of Time

Case 1: The first case in Scenario 3 compared the 2S4P configuration without an IPC connection to 1S8P configuration with each panel connected to an IPC. The difference in output energy can be seen in Table 4.7. The output energy for 1S8P with an IPC is 0.28% (with the 0.8736Ω resistor) more than 2S4P without an IPC. The characteristic curves of PV panels 2S4P and 1S8P are given in Figure 4.14 and 4.15 respectively.

Table 4.7 The Difference in the Output Energy (2S4P and 1S8P)

	Number of Panels in Series	Number of Strings in parallel	Gain	Resistance	The Energy	% difference
No IPC	2	4	N/A	N/A	4446360	N/A
With IPC	N/A	8	10	40	4314960	-2.955
				30	4351800	-2.126
				16	4403280	-0.968
				8.736	4430040	-0.367
				4	4447440	0.024
				2	4454760	0.188
				0.8736	4458840	0.280
				0.6	4459800	0.302
				0.5	4460160	0.310
				0.25	4461120	0.331
				0.1	4461600	0.342

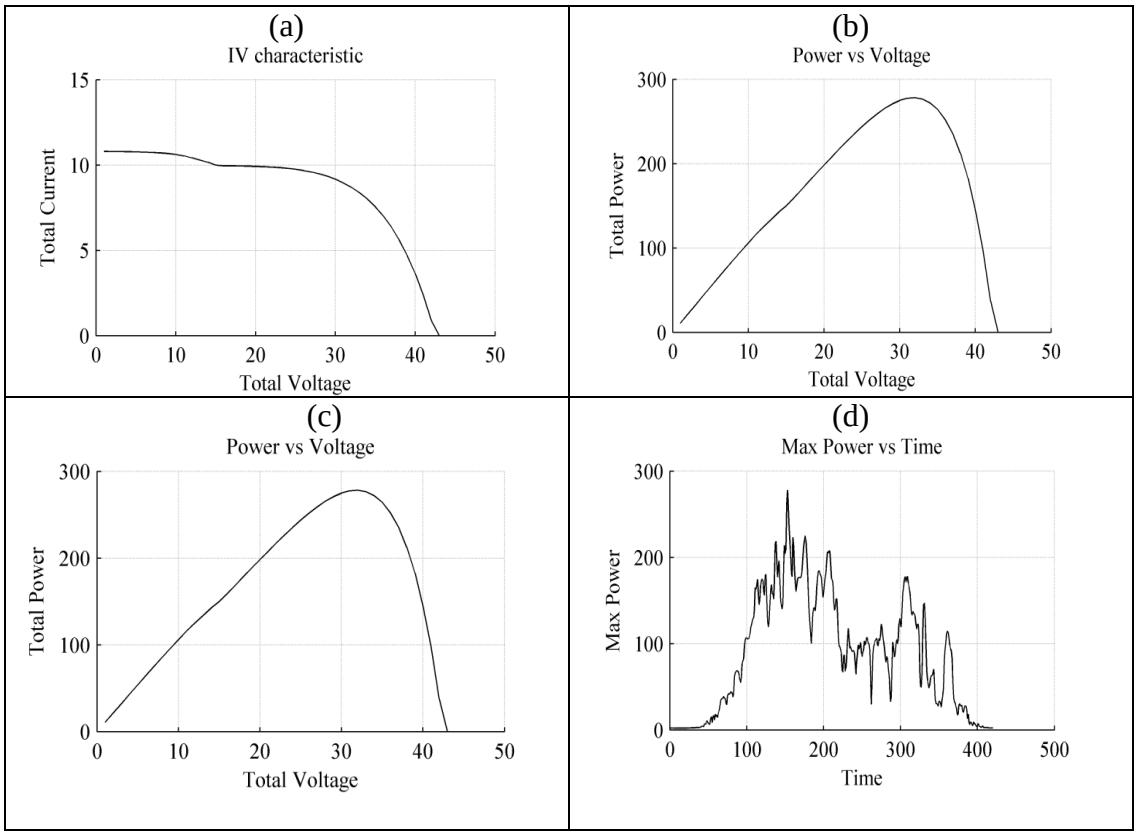


Figure 4.14 Array Characteristics of (2S4P) Without IPC

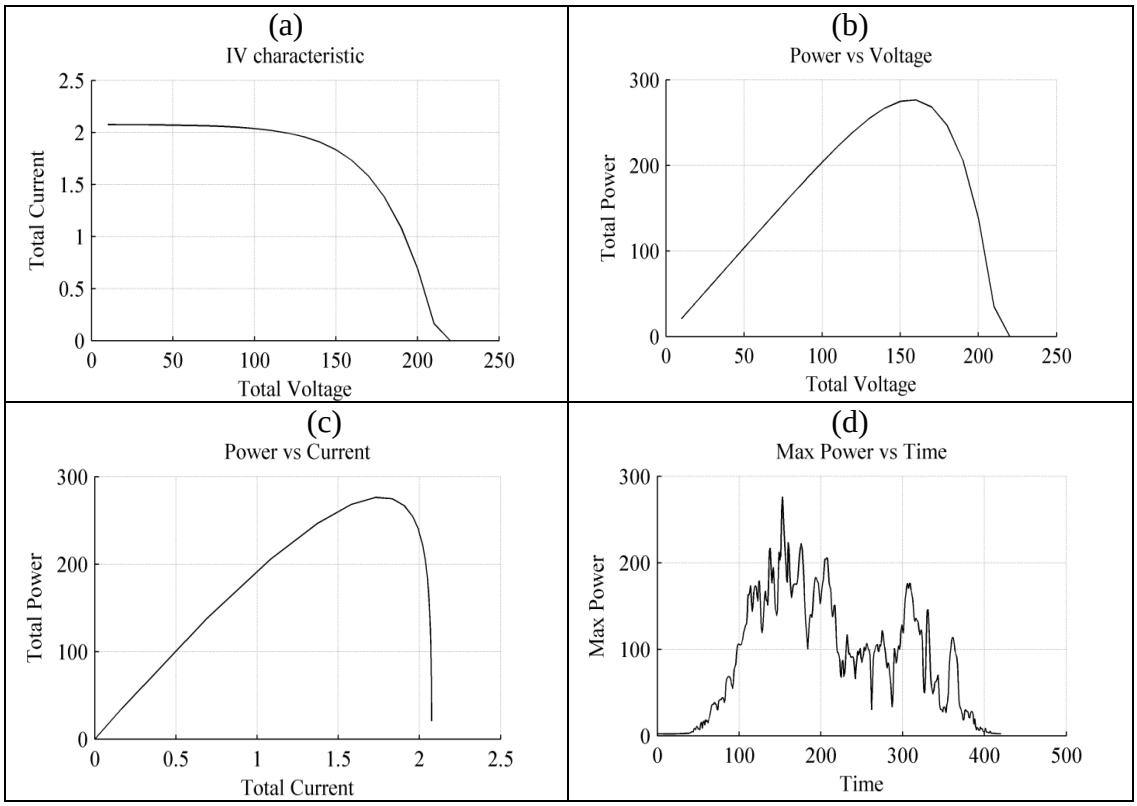


Figure 4.15 Array Characteristics of (1S8P)

Case 2: The second case in Scenario 3 shows the difference between the configuration of a 4S2P without an IPC and a 1S8P with an IPC. The output energy of a 4S2P is less than 2S4P. So, there is more percentage difference in energy between 4S2P and 1S8P compared to 2S4P and 1S8P as shown in Table 4.8. In this scenario 4S2P and 1S8P will result 0.221~0.540% energy difference while 2S4P and 1S8P will result 0.024~0.342% energy difference. The characteristic curves of the PV panel 2S4P can be seen in Figure 4.16.

Table 4.8 The Difference in the Output Energy (4S2P and 1S8P)

	Number of Panels in Series	Number of Strings in parallel	Gain	Resistance	The Energy	% difference
No IPC	4	2	N/A	N/A	4437600	N/A
With IPC	N/A	8	10	40	4314960	-2.763
				30	4351800	-1.933
				16	4403280	-0.773
				8.736	4430040	-0.170
				4	4447440	0.221
				2	4454760	0.386
				0.8736	4458840	0.478
				0.6	4459800	0.500
				0.5	4460160	0.508
				0.25	4461120	0.530
				0.1	4461600	0.540

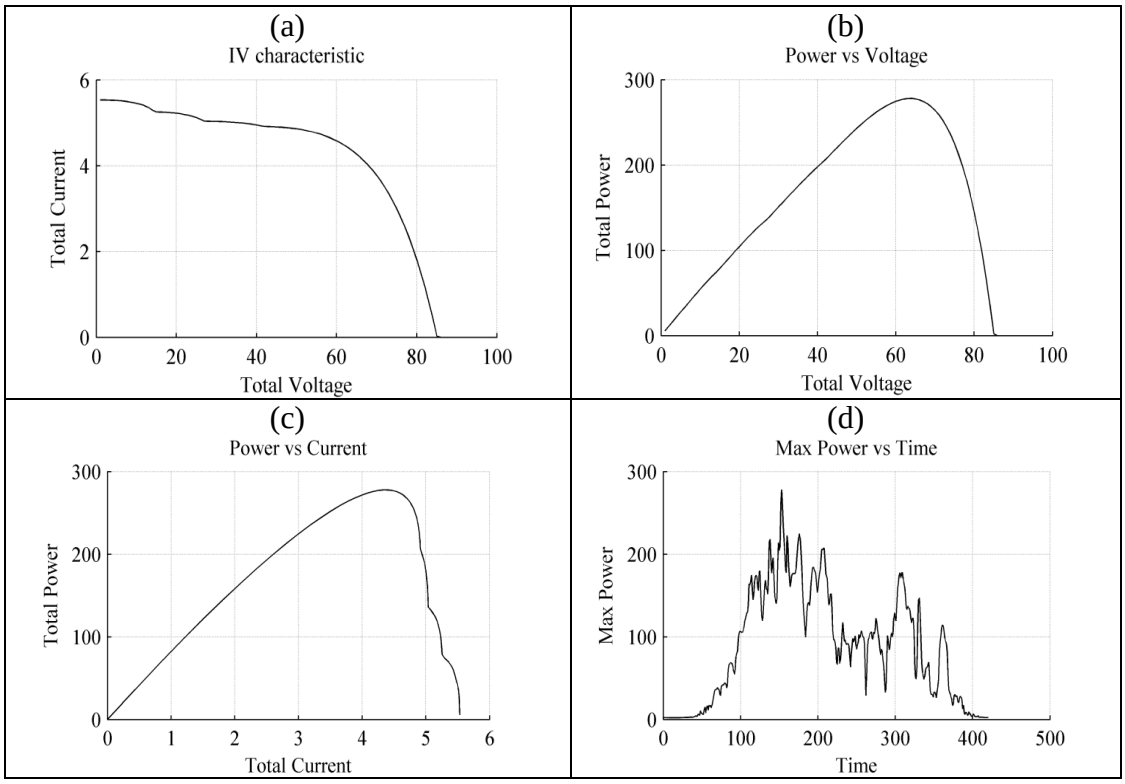


Figure 4.16 Array Characteristics of (4S2P)

Case 3: The third case in Scenario 3 illustrates the difference between the configuration of an 8S1P without an IPC and a1S8P with an IPC. The output energy of a 1S8P is less than that of a 4S2P. So, there is more percentage difference in energy between 8S1P and 1S8P compared to 4S2P and 1S8P as shown in Table 4.9. In this scenario 4S2P and 1S8P will result 0.159~1.372% energy difference while 8S1P and 1S8P will result 0.843~2.065% energy difference. The characteristic curves of the PV panel 2S4P can be seen in Figure 4.17.

Table 4.9 The Difference in the Output Energy (8S1P and 1S8P)

	Number of Panels in Series	**Number of Strings in parallel**	**Gain**	**Resistance**	**The Energy**	**% difference**
No IPC	8	1	N/A	N/A	4434480	N/A
With IPC	N/A	8	10	40	4314960	-2.695
				30	4351800	-1.864
				16	4403280	-0.703
				8.736	4430040	-0.100
				4	4447440	0.292
				2	4454760	0.457
				0.8736	4458840	0.549
				0.6	4459800	0.570
				0.5	4460160	0.579
				0.25	4461120	0.600
				0.1	4461600	0.611

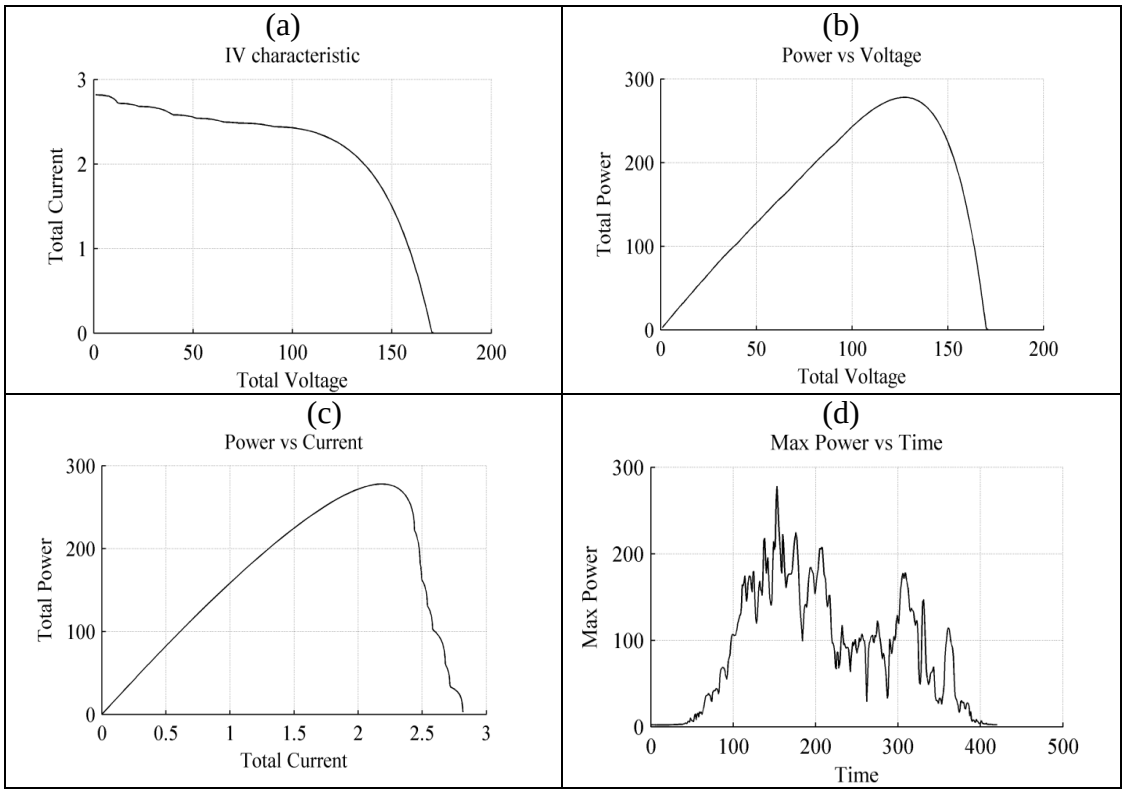


Figure 4.17 Array Characteristics of (8S1P)

The theoretical energy from eight individual panels (if individually connected to MPPTs) was calculated to be 6,446,652 J. Figure 4.18 represents the output energy of the entire configurations: 2S4P, 4S2P, 8S1P and 1S8P divided by the theoretical value. The output energy of a 1S8P is seen decreasing as the resistance of the IPC increases.

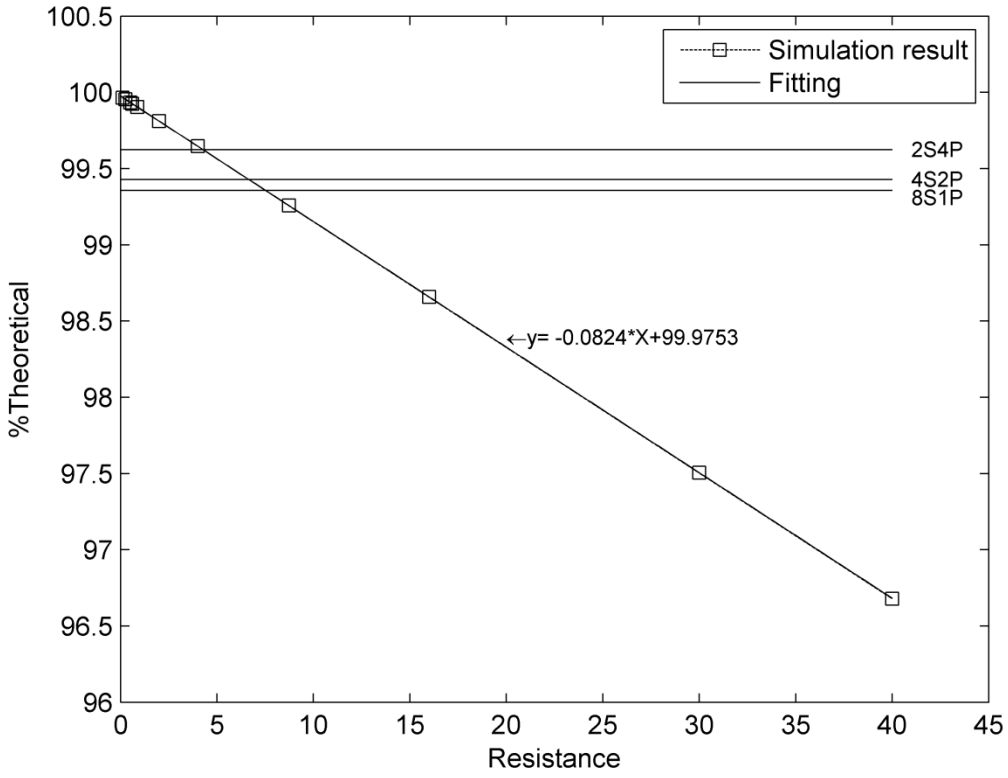


Figure 4.18 %Theoretical vs. Resistance

4.4. SCENARIO 4

The data taken from the sensor array on September 25, 2012 is analyzed in Scenario 4. Figure 4.19 illustrates the radiation measured from the sensors used in the analysis. Panel Uni-Solar 68 was used in the simulation. 532 samples from the sensors were used. Each sample was picked in every 2 minutes.

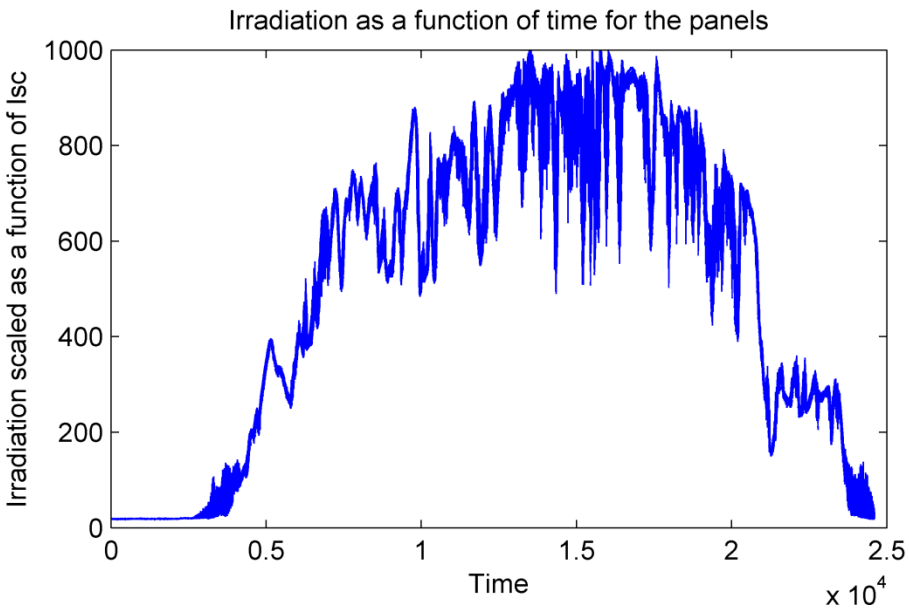


Figure 4.19 The Insolation Data as a Function of Time

Case 1: The first case in Scenario 4 compared the 2S4P configuration without an IPC connection to 1S8P configuration with each panel connected to an IPC. The difference in output energy can be seen in Table 4.10. The output energy for 1S8P with an IPC is 0.296% (with the 0.8736Ω resistor) more than 2S4P without an IPC. The characteristic curves of PV panels 2S4P and 1S8P are given in Figure 4.20 and 4.21 respectively.

Table 4.10 The Difference in the Output Energy (2S4P and 1S8P)

	Number of Panels in Series	Number of Strings in parallel	Gain	Resistance	The Energy	% difference
No IPC	2	4	N/A	N/A	7012560	N/A
With IPC	N/A	8	10	40	6732720	-3.990
				30	6808320	-2.912
				16	6913200	-1.416
				8.736	6970200	-0.604
				4	7008240	-0.061
				2	7024320	0.167
				0.8736	7033320	0.296
				0.6	7035480	0.326
				0.5	7036320	0.338
				0.25	7038360	0.367
				0.1	7039560	0.385

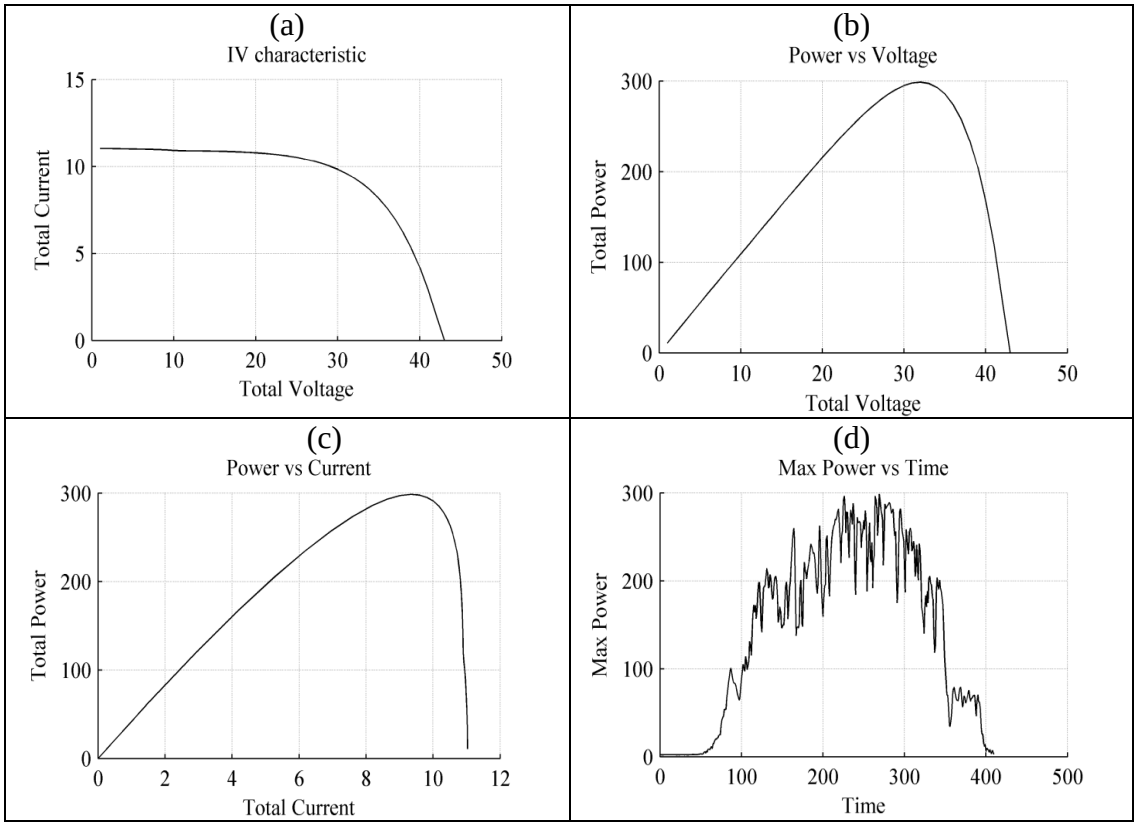


Figure 4.20 Array Characteristics of (2S4P) Without IPC

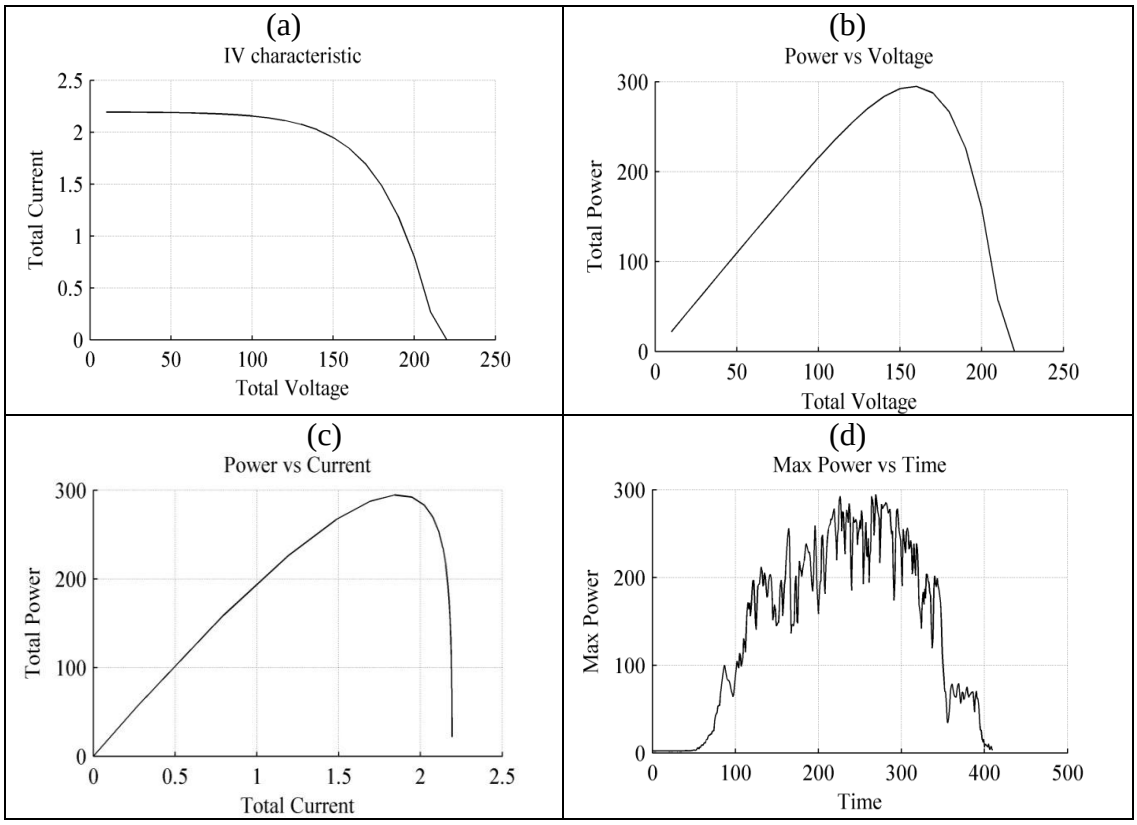


Figure 4.21 Array Characteristics of (1S8P)

Case 2: The second case in Scenario 4 shows the difference between the configuration of a 4S2P without an IPC and a 1S8P with an IPC. The output energy of a 4S2P is less than 2S4P. So, there is more percentage difference in energy between 4S2P and 1S8P compared to 2S4P and 1S8P as shown in Table 4.11. In this scenario 4S2P and 1S8P will result 0.205~0.653% energy difference while 2S4P and 1S8P will result 0.167~0.385% energy difference. The characteristic curves of the PV panel 2S4P can be seen in Figure 4.22.

Table 4.11 The Difference in the Output Energy (4S2P and 1S8P)

	Number of Panels in Series	Number of Strings in parallel	Gain	Resistance	The Energy	% difference
No IPC	4	2	N/A	N/A	6993840	N/A
With IPC	N/A	8	10	40	6732720	-3.733
				30	6808320	-2.652
				16	6913200	-1.153
				8.736	6970200	-0.338
				4	7008240	0.205
				2	7024320	0.435
				0.8736	7033320	0.564
				0.6	7035480	0.595
				0.5	7036320	0.607
				0.25	7038360	0.636
				0.1	7039560	0.653

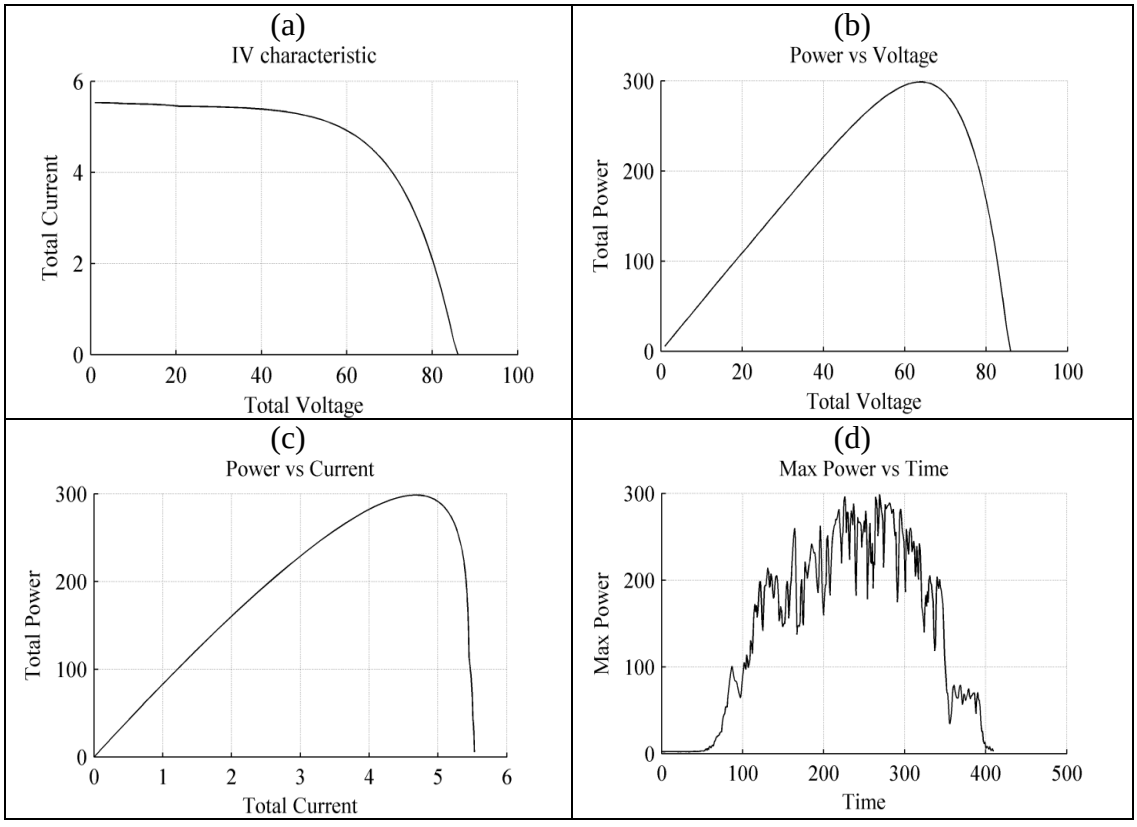


Figure 4.22 Array Characteristics of (4S2P)

Case 3: The third case in Scenario 4 illustrates the difference between the configuration of an 8S1P without an IPC and a 1S8P with an IPC. The output energy of a 1S8P is less than that of a 4S2P. So, there is more percentage difference in energy between 8S1P and 1S8P compared to 4S2P and 1S8P as shown in Table 4.12. In this scenario 4S2P and 1S8P will result 0.205~0.653% energy difference while 8S1P and 1S8P will result 0.393~0.842% energy difference. The characteristic curves of the PV panel 2S4P can be seen in Figure 4.23.

Table 4.12 The Difference in the Output Energy (8S1P and 1S8P)

	Number of Panels in Series	**Number of Strings in parallel**	**Gain**	**Resistance**	**The Energy**	**% difference**
No IPC	8	1	N/A	N/A	6980760	N/A
With IPC	N/A	8	10	40	6732720	-3.553
				30	6808320	-2.470
				16	6913200	-0.967
				8.736	6970200	-0.151
				4	7008240	0.393
				2	7024320	0.624
				0.8736	7033320	0.752
				0.6	7035480	0.783
				0.5	7036320	0.795
				0.25	7038360	0.825
				0.1	7039560	0.842

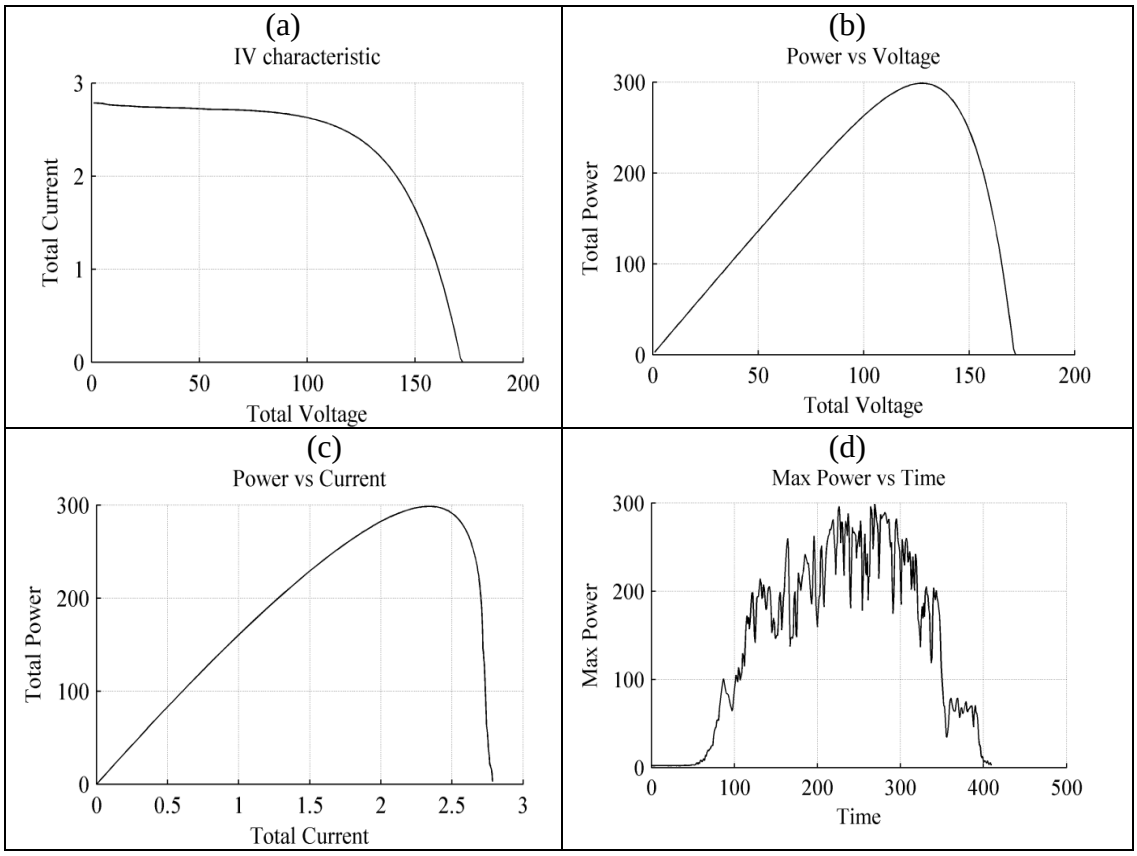


Figure 4.23 Array Characteristics of (8S1P)

The theoretical energy from eight individual panels (if individually connected to MPPTs) was calculated to be 7,042,224 J. Figure 4.24 represents the output energy of the entire configurations: 2S4P, 4S2P, 8S1P and 1S8P divided by the theoretical value. The output energy of a 1S8P is seen decreasing as the resistance of the IPC increases.

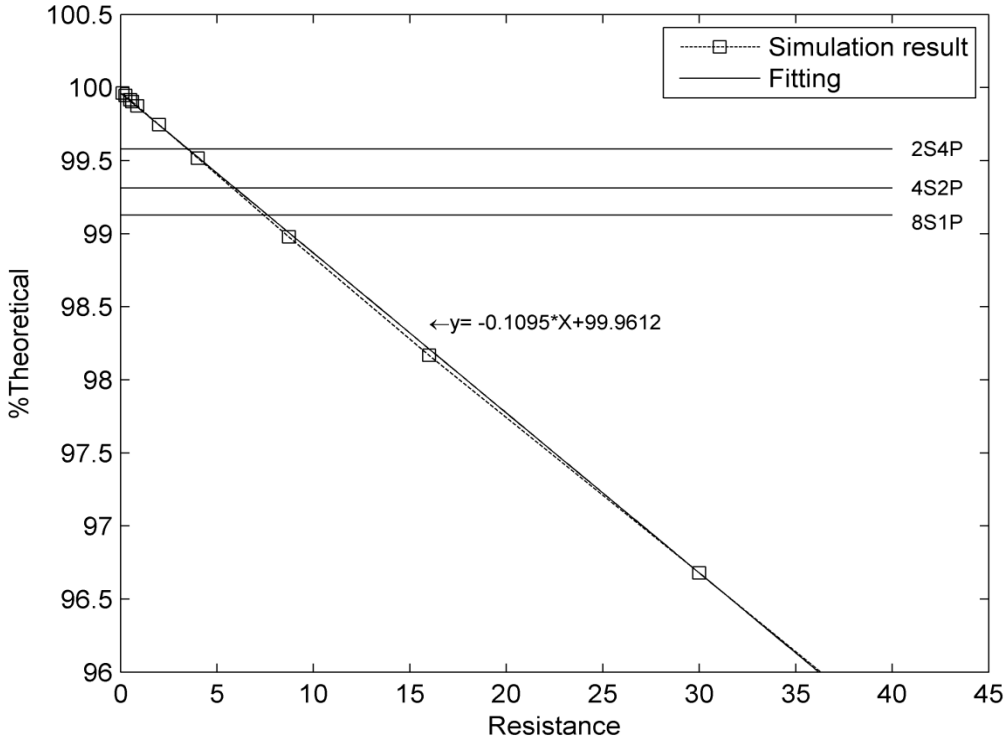


Figure 4.24 %Theoretical vs. Resistance

5. CONCLUSIONS

A comparison between the parallel configured PV array and the conventional series configured PV array is studied in this thesis. The effectiveness of the parallel configuration in terms of the output power obtained was evaluated using software computations. A software package was developed to investigate the advantages of this newly emerging technology over conventional series technology. This software is able to simulate the real system based on the desired configuration.

Simulation results prove that this parallel technique reduced the problem of the existence of multiple MPP in the system, and makes the MPP tracking easier. Using a DC-DC converter with the parallel configuration increases the output voltage of the PV panel and decreases the current which in turn help to reduce the wiring cost. The most important advantage of having such a configuration is that all of the strings share the same voltage that could be easily measured and controlled in order to track the MPP with maximum accuracy. Different shading scenarios have also been analyzed in this thesis to study the shading effect. It was found that in all four scenarios, the parallel configuration worked better under certain shading conditions. Scenario 1 and 2 demonstrated that the parallel configuration resulted 1 to 4% and 0.1 to 2% of increased power output respectively when compared with that of the series configuration. Even when the worst case shading effect scenarios (scenario 3 and 4) are concerned the parallel configuration provided 0.1 to 0.6% and 0.1 to 0.8% more output respectively when compared with that of the series configuration. Also scenario 1 produced 30% more energy than scenario 2. Scenarios 1 and 2 were considered when some of the sensors were shaded and some are

not. Scenarios 3 and 4 consider that all of the sensors were shaded. Thus, the parallel configuration was more effective in the first two scenarios. The typical Energy vs. Resistance performance curves illustrated that the resistance needs to be less than some particular value to provide advantages for parallel connected PV panels. The values found in scenarios 1 and 2 were $25\,\Omega$ and $10\,\Omega$, respectively; the value in scenarios 3 and 4 was $4\,\Omega$. Although the effects of temperature and the tracking process have some impact on output energy, these effects were not taken into account in this software as the temperature data was unavailable.

The use of either the conventional series configuration or the parallel configuration is highly dependent on both the application type and the climatic conditions. There are many researches going on in this area in order to determine the best configuration according to the location, the size of the PV system, and the application types which is beyond the scope of this thesis. A software package such as this can be used as a planning tool for the researchers and the students for understanding different PV array configurations. This tool is useful for training or teaching purpose also.

APPENDIX

MATLAB CODE FOR DEVELOPED SOFTWARE

Main Menu

```
function varargout = MainMenu(varargin)
% MAINMENU MATLAB code for MainMenu.fig
%      MAINMENU, by itself, creates a new MAINMENU or raises the
existing
%      singleton*.
%
%      H = MAINMENU returns the handle to a new MAINMENU or the handle
to
%      the existing singleton*.
%
%      MAINMENU('CALLBACK',hObject,eventData,handles,...) calls the
local
%      function named CALLBACK in MAINMENU.M with the given input
arguments.
%
%      MAINMENU('Property','Value',...) creates a new MAINMENU or
raises the
%      existing singleton*.  Starting from the left, property value
pairs are
%      applied to the GUI before MainMenu_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to MainMenu_OpeningFcn via
varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only
one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help MainMenu

% Last Modified by GUIDE v2.5 13-May-2012 11:54:06

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @MainMenu_OpeningFcn, ...
                   'gui_OutputFcn',  @MainMenu_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{18})
    gui_State.gui_Callback = str2func(varargin{18});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT
% --- Executes just before MainMenu is made visible.
```

```
function MainMenu_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to MainMenu (see VARARGIN)

% Choose default command line output for MainMenu
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes MainMenu wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = MainMenu_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in sccOn.
function sccOn_Callback(hObject, eventdata, handles)
% hObject    handle to sccOn (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of sccOn


% --- Executes on selection change in listbox1.
function listbox1_Callback(hObject, eventdata, handles)
% hObject    handle to listbox1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%  Hints: contents = cellstr(get(hObject,'String')) returns listbox1
contents as cell array
%            contents{get(hObject,'Value')} returns selected item from
listbox1


% --- Executes during object creation, after setting all properties.
function listbox1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to listbox1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles      empty - handles not created until after all CreateFcns
called

% Hint: listbox controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if         ispc         &&         isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on selection change in module.
function module_Callback(hObject, eventdata, handles)
% hObject    handle to module (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%  Hints:  contents  =  cellstr(get(hObject,'String'))  returns  module
contents as cell array
%           contents{get(hObject,'Value')} returns selected item from
module


% --- Executes during object creation, after setting all properties.
function module_CreateFcn(hObject, eventdata, handles)
% hObject    handle to module (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles      empty - handles not created until after all CreateFcns
called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if         ispc         &&         isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function latitude_Callback(hObject, eventdata, handles)
% hObject    handle to latitude (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of latitude as text
%        str2double(get(hObject,'String')) returns contents of latitude
as a double


% --- Executes during object creation, after setting all properties.
function latitude_CreateFcn(hObject, eventdata, handles)
% hObject    handle to latitude (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles      empty - handles not created until after all CreateFcns
called
```

```
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if        ispc        &&        isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function numSeries_Callback(hObject, eventdata, handles)
% hObject    handle to numSeries (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of numSeries as text
%               str2double(get(hObject,'String'))  returns  contents  of
numSeries as a double


% --- Executes during object creation, after setting all properties.
function numSeries_CreateFcn(hObject, eventdata, handles)
% hObject    handle to numSeries (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles     empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if        ispc        &&        isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function numParallel_Callback(hObject, eventdata, handles)
% hObject    handle to numParallel (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of numParallel as text
%               str2double(get(hObject,'String'))  returns  contents  of
numParallel as a double


% --- Executes during object creation, after setting all properties.
function numParallel_CreateFcn(hObject, eventdata, handles)
% hObject    handle to numParallel (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles     empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
```

```
if        ispc        &&        isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function gain_Callback(hObject, eventdata, handles)
% hObject    handle to gain (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of gain as text
%        str2double(get(hObject,'String')) returns contents of gain as
a double


% --- Executes during object creation, after setting all properties.
function gain_CreateFcn(hObject, eventdata, handles)
% hObject    handle to gain (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles     empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if        ispc        &&        isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function resistance_Callback(hObject, eventdata, handles)
% hObject    handle to resistance (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of resistance as text
%               str2double(get(hObject,'String'))  returns  contents  of
resistance as a double


% --- Executes during object creation, after setting all properties.
function resistance_CreateFcn(hObject, eventdata, handles)
% hObject    handle to resistance (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles     empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if        ispc        &&        isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
```

```
end


% --- Executes on selection change in day.
function day_Callback(hObject, eventdata, handles)
% hObject    handle to day (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns day contents
as cell array
%        contents{get(hObject,'Value')} returns selected item from day


% --- Executes during object creation, after setting all properties.
function day_CreateFcn(hObject, eventdata, handles)
% hObject    handle to day (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles      empty - handles not created until after all CreateFcns
called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if         ispc         &&         isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function year_Callback(hObject, eventdata, handles)
% hObject    handle to year (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of year as text
%         str2double(get(hObject,'String')) returns contents of year as
a double


% --- Executes during object creation, after setting all properties.
function year_CreateFcn(hObject, eventdata, handles)
% hObject    handle to year (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles      empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if         ispc         &&         isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
% --- Executes on selection change in month.
function month_Callback(hObject, eventdata, handles)
% hObject    handle to month (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%  Hints: contents = cellstr(get(hObject,'String')) returns month
contents as cell array
%         contents{get(hObject,'Value')} returns selected item from
month


% --- Executes during object creation, after setting all properties.
function month_CreateFcn(hObject, eventdata, handles)
% hObject    handle to month (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in buttonOK.
function buttonOK_Callback(hObject, eventdata, handles)
% hObject    handle to buttonOK (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

InsolationMenu();


% --- Executes when selected object is changed in sccSelect.
function sccSelect_SelectionChangeFcn(hObject, eventdata, handles)
% hObject    handle to the selected object in sccSelect
% eventdata  structure with the following fields (see UIBUTTONGROUP)
%   EventName: string 'SelectionChanged' (read only)
%   OldValue: handle of the previously selected object or empty if none
was selected
%   NewValue: handle of the currently selected object
% handles    structure with handles and user data (see GUIDATA)
handles = guidata(hObject);

switch get(eventdata.NewValue,'Tag')   % Get Tag of selected object


    case 'sccOn'
        set(handles.numSeries, 'Enable', 'off');
        set(handles.gain, 'Enable', 'on');
        set(handles.resistance, 'Enable', 'on')
```

```
    case 'sccOff'
        set(handles.numSeries, 'Enable', 'on');
        set(handles.gain, 'Enable', 'off');
        set(handles.gain, 'String', 1);
        set(handles.resistance, 'Enable', 'off');
        set(handles.resistance, 'String', 0);

    otherwise


       % Code for when there is no match.


end
%updates the handles structure
guidata(hObject, handles);


% --- Executes during object creation, after setting all properties.
function sccSelect_CreateFcn(hObject, eventdata, handles)
% hObject    handle to sccSelect (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles     empty - handles not created until after all CreateFcns
called
```

Insolation Menu

```
function varargout = InsolationMenu(varargin)
% INSOLATIONMENU MATLAB code for InsolationMenu.fig
%         INSOLATIONMENU, by itself, creates a new INSOLATIONMENU or
raises the existing
%      singleton*.
%
%      H = INSOLATIONMENU returns the handle to a new INSOLATIONMENU or
the handle to
%      the existing singleton*.
%
%        INSOLATIONMENU('CALLBACK',hObject,eventData,handles,...) calls
the local
%      function named CALLBACK in INSOLATIONMENU.M with the given input
arguments.
%
%              INSOLATIONMENU('Property','Value',...)  creates  a  new
INSOLATIONMENU or raises the
%        existing singleton*.  Starting from the left, property value
pairs are
%      applied to the GUI before InsolationMenu_OpeningFcn gets called.
An
%         unrecognized property name or invalid value makes property
application
%       stop.  All inputs are passed to InsolationMenu_OpeningFcn via
varargin.
%
```

```
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only
one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help InsolationMenu

% Last Modified by GUIDE v2.5 11-May-2012 15:13:57

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @InsolationMenu_OpeningFcn, ...
                   'gui_OutputFcn',  @InsolationMenu_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before InsolationMenu is made visible.
function    InsolationMenu_OpeningFcn(hObject,    eventdata,    handles,
varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to InsolationMenu (see VARARGIN)

% Choose default command line output for InsolationMenu
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes InsolationMenu wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function   varargout   =   InsolationMenu_OutputFcn(hObject,   eventdata,
handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


function boxMin_Callback(hObject, eventdata, handles)
% hObject    handle to boxMin (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of boxMin as text
%          str2double(get(hObject,'String')) returns contents of boxMin
as a double


% --- Executes during object creation, after setting all properties.
function boxMin_CreateFcn(hObject, eventdata, handles)
% hObject    handle to boxMin (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles     empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if        ispc        &&        isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function boxMax_Callback(hObject, eventdata, handles)
% hObject    handle to boxMax (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of boxMax as text
%          str2double(get(hObject,'String')) returns contents of boxMax
as a double


% --- Executes during object creation, after setting all properties.
function boxMax_CreateFcn(hObject, eventdata, handles)
% hObject    handle to boxMax (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles     empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if        ispc        &&        isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
% --- Executes on button press in buttonBrowse.
function buttonBrowse_Callback(hObject, eventdata, handles)
% hObject    handle to buttonBrowse (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%gets input file(s) from user
[input_file,pathname] = uigetfile( ...
       {'*.csv', 'CSV (*.csv)'; ...
       '*.xls', 'Excel (*.xls)';...
        '*.*', 'All Files (*.*)'}, ...
        'Select files', ...
        'MultiSelect', 'on');

%if file selection is cancelled, pathname should be zero
%and nothing should happen
if pathname == 0
    return
end

%gets the current data file names inside the listbox
inputFileNames = get(handles.boxPath,'String');

%if they only select one file, then the data will not be a cell
%if more than one file selected at once,
%then the data is stored inside a cell
if iscell(input_file) == 0

    %add the most recent data file selected to the cell containing
    %all the data file names
    inputFileNames{end+1} = fullfile(pathname,input_file);

%else, data will be in cell format
else
    %stores full file path into inputFileNames
    for n = 1:length(input_file)
        %notice the use of {}, because we are dealing with a cell here!
        inputFileNames{end+1} = fullfile(pathname,input_file{n});
    end
end

%updates the gui to display all filenames in the listbox
set(handles.boxPath,'String',inputFileNames);

%make sure first file is always selected so it doesn't go out of range
%the GUI will break if this value is out of range
set(handles.boxPath,'Value',1);
% Update handles structure
guidata(hObject, handles);


% --- Executes on button press in buttonOK.
function buttonOK_Callback(hObject, eventdata, handles)
% hObject    handle to buttonOK (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)

dataSource=get(handles.choice,'Value');

if(get(handles.choice, 'Value') == 0)
    path=get(handles.boxPath,'String');
elseif(get(handles.choice, 'Value') == 1)
    min=get(handles.boxMin,'String');
    max=get(handles.boxMax,'String');
elseif(get(handles.choice, 'Value') == 2)
    ManualData();
end

PowerCalculations();


% --- Executes when selected object is changed in dataSourceSelect.
function     dataSourceSelect_SelectionChangeFcn(hObject,     eventdata,
handles)
handles = guidata(hObject);

switch get(eventdata.NewValue,'Tag')   % Get Tag of selected object

    case 'dataFile'
        set(handles.choice, 'String', '0');
        set(handles.buttonBrowse,'Visible', 'on');
        set(handles.textMax, 'Visible', 'off');
        set(handles.textMin, 'Visible', 'off');
        set(handles.textFile,'Visible','on');
        set(handles.textRandom,'Visible','off');
        set(handles.boxMax,'Visible','off');
        set(handles.boxMin,'Visible','off');
        set(handles.boxPath,'Visible','on');
        set(handles.buttonBrowse,'Enable', 'on');
        set(handles.boxPath,'Enable', 'on');
        set(handles.boxMin,'Enable', 'off');
        set(handles.boxMax,'Enable', 'off');

    case 'dataRandom'
        set(handles.choice, 'String', '1');
        set(handles.buttonBrowse,'Visible', 'off');
        set(handles.textMax, 'Visible', 'on');
        set(handles.textMin, 'Visible', 'on');
        set(handles.textFile,'Visible','off');
        set(handles.textRandom,'Visible','on');
        set(handles.boxMax,'Visible','on');
        set(handles.boxMin,'Visible','on');
        set(handles.boxPath,'Visible','off');
        set(handles.buttonBrowse, 'Enable', 'off');
        set(handles.boxPath,'Enable', 'off');
        set(handles.boxMin,'Enable', 'on');
        set(handles.boxMax,'Enable', 'on');

    case 'dataManual'
        set(handles.choice, 'String', '2');
```

```
        set(handles.buttonBrowse,'Visible', 'off');
        set(handles.textMax, 'Visible', 'off');
        set(handles.textMin, 'Visible', 'off');
        set(handles.textFile,'Visible','off');
        set(handles.textRandom,'Visible','off');
        set(handles.boxMax,'Visible','off');
        set(handles.boxMin,'Visible','off');
        set(handles.boxPath,'Visible','off');
        set(handles.buttonBrowse,'Enable', 'off');
        set(handles.boxPath,'Enable', 'off');
        set(handles.boxMin,'Enable', 'off');
        set(handles.boxMax,'Enable', 'off');

    otherwise
        set(handles.choice, 'String', '0');

end

guidata(hObject, handles);
```

InputMatrix

```
function varargout = InputMatrix(varargin)
% INPUTMATRIX M-file for InputMatrix.fig
%       INPUTMATRIX, by itself, creates a new INPUTMATRIX or raises the
existing
%      singleton*.
%
%       H = INPUTMATRIX returns the handle to a new INPUTMATRIX or the
handle to
%      the existing singleton*.
%
%       INPUTMATRIX('CALLBACK',hObject,eventData,handles,...) calls the
local
%       function named CALLBACK in INPUTMATRIX.M with the given input
arguments.
%
%      INPUTMATRIX('Property','Value',...) creates a new INPUTMATRIX or
raises the
%        existing singleton*.  Starting from the left, property value
pairs are
%       applied to the GUI before InputMatrix_OpeningFcn gets called.
An
%         unrecognized property name or invalid value makes property
application
%         stop.   All inputs are passed to InputMatrix_OpeningFcn via
varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only
one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help InputMatrix
```

```
% Last Modified by GUIDE v2.5 13-May-2012 18:37:48

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @InputMatrix_OpeningFcn, ...
                   'gui_OutputFcn',  @InputMatrix_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before InputMatrix is made visible.
function InputMatrix_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to InputMatrix (see VARARGIN)

% Choose default command line output for InputMatrix
handles.output = hObject;


set(handles.data, 'data',varargin{1});


% Update handles structure
guidata(hObject, handles);

% UIWAIT makes InputMatrix wait for user response (see UIRESUME)
uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = InputMatrix_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = get(handles.data,'data');
% The figure can be deleted now
```

```
delete(handles.figure1);



% --- Executes when user attempts to close figure1.
function figure1_CloseRequestFcn(hObject, eventdata, handles)
% hObject    handle to figure1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: delete(hObject) closes the figure
if isequal(get(hObject, 'waitstatus'), 'waiting')
    % The GUI is still in UIWAIT, us UIRESUME
    uiresume(hObject);
else
    % The GUI is no longer waiting, just close it
    delete(hObject);
end



% --- Executes on button press in ok.
function ok_Callback(hObject, eventdata, handles)
% hObject    handle to ok (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
close(handles.figure1);
```

Random

```
function varargout = random(varargin)
% RANDOM M-file for random.fig
%      RANDOM, by itself, creates a new RANDOM or raises the existing
%      singleton*.
%
%      H = RANDOM returns the handle to a new RANDOM or the handle to
%      the existing singleton*.
%
%      RANDOM('CALLBACK',hObject,eventData,handles,...) calls the local
%          function named CALLBACK in RANDOM.M with the given input
arguments.
%
%        RANDOM('Property','Value',...) creates a new RANDOM or raises
the
%        existing singleton*.  Starting from the left, property value
pairs are
%      applied to the GUI before random_OpeningFcn gets called.  An
%         unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to random_OpeningFcn via varargin.
%
%       *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only
one
%      instance to run (singleton)".
%
```

```
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help random

% Last Modified by GUIDE v2.5 28-Dec-2010 10:38:26

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @random_OpeningFcn, ...
                   'gui_OutputFcn',  @random_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before random is made visible.
function random_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to random (see VARARGIN)

% Choose default command line output for random
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes random wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = random_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;
```

```
% --- Executes on slider movement.
function slider1_Callback(hObject, eventdata, handles)
% hObject    handle to slider1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%          get(hObject,'Min') and get(hObject,'Max') to determine range
of slider


% --- Executes during object creation, after setting all properties.



% --- Executes on selection change in listbox.
function listbox_Callback(hObject, eventdata, handles)
% hObject    handle to listbox (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%  Hints: contents = cellstr(get(hObject,'String')) returns listbox
contents as cell array
%           contents{get(hObject,'Value')} returns selected item from
listbox


% --- Executes during object creation, after setting all properties.
function listbox_CreateFcn(hObject, eventdata, handles)
% hObject    handle to listbox (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles     empty - handles not created until after all CreateFcns
called

% Hint: listbox controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if        ispc        &&        isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

Ifcustom

```
function varargout = ifcustom(varargin)
% IFCUSTOM M-file for ifcustom.fig
%         IFCUSTOM, by itself, creates a new IFCUSTOM or raises the
existing
%      singleton*.
%
%       H = IFCUSTOM returns the handle to a new IFCUSTOM or the handle
to
%      the existing singleton*.
%
%          IFCUSTOM('CALLBACK',hObject,eventData,handles,...) calls the
local
```

```
%         function named CALLBACK in IFCUSTOM.M with the given input
arguments.
%
%          IFCUSTOM('Property','Value',...) creates a new IFCUSTOM or
raises the
%        existing singleton*.  Starting from the left, property value
pairs are
%      applied to the GUI before ifcustom_OpeningFcn gets called.  An
%         unrecognized property name or invalid value makes property
application
%          stop.   All inputs are passed to ifcustom_OpeningFcn via
varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only
one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help ifcustom

% Last Modified by GUIDE v2.5 13-May-2012 13:52:57

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @ifcustom_OpeningFcn, ...
                   'gui_OutputFcn',  @ifcustom_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before ifcustom is made visible.
function ifcustom_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to ifcustom (see VARARGIN)

% Choose default command line output for ifcustom
handles.output = hObject;

% Update handles structure
```

```
guidata(hObject, handles);

% UIWAIT makes ifcustom wait for user response (see UIRESUME)
uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = ifcustom_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
temporary = zeros(1,6);
temporary(1) = str2double(get(handles.Isc, 'String'));
temporary(2) = str2double(get(handles.Vmax, 'String'));
temporary(3) = str2double(get(handles.Imax, 'String'));
temporary(4) = str2double(get(handles.Voc, 'String'));
temporary(5) = str2double(get(handles.numCells, 'String'));
temporary(6) = str2double(get(handles.ratedPower, 'String'));
varargout{1} = temporary;
% The figure can be deleted now
delete(handles.figure1);




% --- Executes when user attempts to close figure1.
function figure1_CloseRequestFcn(hObject, eventdata, handles)
% hObject    handle to figure1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: delete(hObject) closes the figure
if isequal(get(hObject, 'waitstatus'), 'waiting')
    % The GUI is still in UIWAIT, us UIRESUME
    uiresume(hObject);
else
    % The GUI is no longer waiting, just close it
    delete(hObject);
end



% --- Executes on button press in ok.
function ok_Callback(hObject, eventdata, handles)
% hObject    handle to ok (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
close(handles.figure1);


function moduleName_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to moduleName (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of moduleName as text
%               str2double(get(hObject,'String'))  returns  contents  of
moduleName as a double


% --- Executes during object creation, after setting all properties.
function moduleName_CreateFcn(hObject, eventdata, handles)
% hObject    handle to moduleName (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles     empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if         ispc         &&         isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function Isc_Callback(hObject, eventdata, handles)
% hObject    handle to Isc (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Isc as text
%        str2double(get(hObject,'String')) returns contents of Isc as a
double


% --- Executes during object creation, after setting all properties.
function Isc_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Isc (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles     empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if         ispc         &&         isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function Vmax_Callback(hObject, eventdata, handles)
% hObject    handle to Vmax (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Vmax as text
%        str2double(get(hObject,'String')) returns contents of Vmax as
a double


% --- Executes during object creation, after setting all properties.
function Vmax_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Vmax (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if        ispc        &&        isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function Imax_Callback(hObject, eventdata, handles)
% hObject    handle to ratedPower (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of ratedPower as text
%               str2double(get(hObject,'String'))  returns  contents  of
ratedPower as a double


% --- Executes during object creation, after setting all properties.
function Imax_CreateFcn(hObject, eventdata, handles)
% hObject    handle to ratedPower (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if        ispc        &&        isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end




function Voc_Callback(hObject, eventdata, handles)
% hObject    handle to ratedPower (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hints: get(hObject,'String') returns contents of ratedPower as text
%               str2double(get(hObject,'String'))  returns  contents  of
ratedPower as a double


% --- Executes during object creation, after setting all properties.
function Voc_CreateFcn(hObject, eventdata, handles)
% hObject    handle to ratedPower (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles     empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if        ispc        &&        isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function numCells_Callback(hObject, eventdata, handles)
% hObject    handle to Voc (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Voc as text
%        str2double(get(hObject,'String')) returns contents of Voc as a
double


% --- Executes during object creation, after setting all properties.
function numCells_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Voc (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles     empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if        ispc        &&        isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function ratedPower_Callback(hObject, eventdata, handles)
% hObject    handle to numCells (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of numCells as text
```

```
%        str2double(get(hObject,'String')) returns contents of numCells
as a double


% --- Executes during object creation, after setting all properties.
function ratedPower_CreateFcn(hObject, eventdata, handles)
% hObject    handle to numCells (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if        ispc        &&        isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

Irradiation

```
function [ irradiationIscSystem Temperature ] = Irradiation(dataSource,
ns , np, filePath, ti , minRange, maxRange)

numPanels =  ns * np;
% ti = Number of datapoints to use per module.
% n  = Number of datapoints neglected per module.

irradiationIscSystem = zeros(ti,numPanels);
Temperature = 25 * ones(1,numPanels);

if(dataSource == 0)
    R = 2;      %2 Row to begin reading file
    C = 1;          % Column to begin reading file
    n=60;           % Number of datapoints neglected per module

    figure
    All_irradiationIscSystem = zeros(n*ti,numPanels);

    for x = 1:length(filePath)
        filePath[14];
    end

    for j=1:numPanels
        irradiation=csvread(filePath[14],R,C,[R C (ti*n)+2 C]);
        % Need to implement correct scale factor
        % irradiationIsc=irradiation/(0.1(Rsensor*(1+R2/R1)))
        % Rsensor= 1.2, R1=392, R2=6.8 K
        irradiationIsc=irradiation/2.201632653061224;

        irradiationIscMax=max(irradiationIsc);
        irradiationScaled=irradiationIsc/irradiationIscMax*1000;

        C=C+1;

        for p=1:(ti*n)
```

```
        All_irradiationIscSystem(p,j)=irradiationIsc(p);   %   need   to
implement correct scale factor

        end
        % Plots the insolation for each panel as a function of Isc
        k=1:length(irradiationScaled);
        plot(k,irradiationScaled);
        hold on
        set(gcf,'Position',[100 100 500 300])
        title('Irradiation as a function of time for the panels')
        ylabel('Irradiation scaled as a function of Isc')
        xlabel('Time')
    end

    irradiationIscSystem=All_irradiationIscSystem(1:n:end,:)

elseif(dataSource == 1)

    matrix = minRange + (maxRange-minRange).*rand(ns,np);

    %display in random figure file
    randomfigurehandle=random;
    random_data=guidata(randomfigurehandle);
    insol_str=num2str(matrix*1000);
    set(random_data.listbox, 'String', insol_str);

    w=1;
    for j=1:1:ns
        for k=1:1:np
            irradiationIscSystem(w) = matrix(j,k);
            w = w+1;
        end
    end

elseif(dataSource == 2)
    matrix = InputMatrix(zeros(ns, np));

    w=1;
    for j=1:1:ns
        for k=1:1:np
            irradiationIscSystem(w) = matrix(j,k);
            w = w+1;
        end
    end

end
```

Power Calculations

```
%% Import data and close GUI files

MainMenufigurehandle = MainMenu;
MainMenu_data = guidata(MainMenufigurehandle);
    sccOn = get(MainMenu_data.sccOn, 'value');
    numSeriesStr = get(MainMenu_data.numSeries, 'string');
```

```
if(isempty(numSeriesStr) == 1 )
    numSeries = 1;
else
    numSeries = str2double(numSeriesStr);
end
numParallelStr = get(MainMenu_data.numParallel, 'string');
numParallel = str2double(numParallelStr);
gainStr = get(MainMenu_data.gain, 'string');
if(isempty(gainStr) == 1 )
    gain = 1;
else
    gain = str2double(gainStr);
end
resistanceStr = get(MainMenu_data.resistance, 'string');
if(isempty(resistanceStr) == 1 )
    resistance = 0;
else
    resistance = str2double(resistanceStr);
end
latitudeStr = get(MainMenu_data.latitude, 'string');
latitude = str2double(latitudeStr);
day = get(MainMenu_data.day, 'value');
month = get(MainMenu_data.month, 'value');
yearStr = get(MainMenu_data.year, 'string');
year = str2double(yearStr);


switch get(MainMenu_data.module, 'value')
    case 1 % UniSolar 68
        moduleName = 'UniSolar 68';
        Isc = 5.1;
        Vmax = 16.5;
        Imax = 4.1;
        Voc = 23.1;
        numCells = 11;
        ratedPower = 68;

        A =   9.5913;
        Rs =  0.1115;
        Rsh = 4.9975e+03;

    case 2 % BP Solar 75
        moduleName = 'BP Solar 75';
        Isc = 4.7;
        Vmax = 17.3;
        Imax = 4.3;
        Voc = 21.8;
        numCells = 36;
        ratedPower = 75;

        A =  0.9173;
        Rs =0.3643;
        Rsh = 1.5382e+03;

    case 3 % Sun Electronics 120;
        moduleName = 'Sun Electronics 120';
```

```
            Isc = 7.62;
            Vmax = 17.27;
            Imax = 6.95;
            Voc = 21.34;
            numCells = 45;
            ratedPower = 120;

            A =   0.6154;
            Rs = 0.2390;
            Rsh =3.4310e+03;

        case 4 % SunPower 215
            moduleName = 'SunPower 215';
            Isc = 5.8;
            Vmax = 39.8;
            Imax = 5.4;
            Voc = 48.3;
            numCells = 72;
            ratedPower = 215;

            A =  0.9700;
            Rs =0.3298;
            Rsh = 4.1929e+03;


        case 5 % Kyocera 215
            moduleName = 'Kyocera 21';
            Isc = 8.78;
            Vmax = 26.6;
            Imax = 8.09;
            Voc = 33.2;
            numCells = 54;
            ratedPower = 215;

            A =  0.8121;
            Rs = 0.3221;
            Rsh =4.0084e+03;


        case 6 %call custom screen
            %This is if the user wants to enter his own
            %paramters from a datasheet. this calls the 'Ifcustom' GUI
where
            %the user can enter the values
            %[Isc , Vmax , Voc , numCells , ratedPower] = ifcustom();
            constants = ifcustom();
            Isc = constants(1);
            Vmax = constants(2);
            Imax = constants(3);
            Voc = constants(4);
            numCells = constants(5);
            ratedPower = constants(6);

            A = 1.0801;
            Rs = 1.1314;
```

```
            Rsh = 15.3896;

%%%%%%NEEDS                              TO                                  BE
ADDED%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    end
close(MainMenu);

minRange = 0;
maxRange = 0;
ti = 0;
InsolationMenufigurehandle = InsolationMenu;
InsolationMenu_data = guidata(InsolationMenufigurehandle);
    dataSource = 0;
    dataFileSelection = get(InsolationMenu_data.dataFile, 'value');
    filePath = 'none';
    if(dataFileSelection == 1)
        dataSource = 0;
        ti = 532; % Number of datapoints to use per module
        filePath = get(InsolationMenu_data.boxPath, 'string');
    else
        dataRandomSelection     =     get(InsolationMenu_data.dataRandom,
'value');
        if(dataRandomSelection == 1)
            dataSource = 1;
            ti = 1;
            minRangeStr = get(InsolationMenu_data.boxMin, 'string');
            maxRangeStr = get(InsolationMenu_data.boxMax, 'string');
            minRange = str2double(minRangeStr);
            maxRange = str2double(maxRangeStr);
        else
            dataManualSelection   =   get(InsolationMenu_data.dataManual,
'value');
            if(dataManualSelection == 1)
                dataSource = 2;
                ti = 1;
            end
        end
    end
close(InsolationMenu);

%% Get irradiation data
numPanels = numParallel*numSeries; % Number of panels

[   sysIrrad,   Temperature   ]   =   Irradiation(dataSource,   numSeries,
numParallel, filePath, ti, minRange, maxRange);

%%
%!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!
VocTempCo = 0;
IscTempCo = 0;
vStringTotal = 0;
TBD=0;

%!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!
```

```
%% Calculate current and power
panel = 1;
np = 1;
ns = 1;
while panel <= numPanels

    for voltage = 1:1:round(Voc)
        PmaxTotal = zeros(1:length(voltage));
        while np <= numParallel

            for current = 1:.1:(round(Isc * 10)) / 10
                stringPower = zeros(length(current));
                while ns <= numSeries
                    realVoc    =    Voc    -    (VocTempCo    *    (25    -
Temperature(panel)));

                    realIscTemp = Isc * (1 + IscTempCo);
                    realIsc = realIscTemp * (sysIrrad(panel) / 1000);

                    Pmax = Vmax * Imax;
                    realPmaxTemp    =    Pmax    *    (1-.005    *    (25    -
Temperature(panel)));

                    if vStringTotal > (voltage + 1) || vStringTotal <
(voltage - 1)
                        stringPower(current) = TBD;
                    else
                        stringPower(current) = 0;
                    end
                    ns = ns + 1;
                    panel = panel + 1;
                end
                %add module powers here
            end

            PmaxTotal(voltage) = Pmax;
            np = np + 1;
        end
        panel = panel + 1;
    end

end


%%
v_size = floor(Voc*numSeries);
one_row_sysIrrad = zeros(1,v_size);
i = zeros(ti, v_size);
p = zeros(ti, v_size);
one_column_i = zeros(1,ti);
one_column_p = zeros(1,ti);
i_max = zeros(1,v_size);
p_max = zeros(1,v_size);
max_p_in_row = zeros(1,ti);
```

```
global rows_index;

for rows_index=1:ti


    disp(['processing row ' num2str(rows_index)]);

    for index=1:1:numPanels
        one_row_sysIrrad(index)=sysIrrad(rows_index,index);
    end

    % should use the constances after the correction !!!
    parameters      =   SetParameters   (resistance,   gain,   numPanels,
one_row_sysIrrad, Isc, Voc, numCells, A, Rs, Rsh);

    v = 1:1:Voc*numSeries;
    one_row_i = zeros(size(v));
    size_v_temp=size(v);
    size_v=size_v_temp(:,2);
    for k=1:length(v)
        vex = v(k); % external voltage
        %%%%vpanel = vex / ns; % initial cell voltage
        temp = 0;
        % find current
        for j = 1:numParallel
            params_used   =   parameters(   (j-1)*numSeries   +   1   :   (j-
1)*numSeries+numSeries,: );
            current  =  SeriesCalculations(  vex,  numSeries,  params_used
);
            if current >0
                temp = temp + current;
            end
        end
        one_row_i(k) = temp;
    end
    one_row_p = one_row_i.*v;


     max_p_in_row(rows_index) = max(one_row_p);

    for g=1:size_v
        i(rows_index,g)=one_row_i(g);
        p(rows_index,g)=one_row_p(g);
    end

   end


%%
% agrugate the values

for j=1:size_v
    for k=1:ti
        one_column_i(k)=i(k,j);
```

```
        one_column_p(k)=p(k,j);
    end
    i_max(j)=max(one_column_i);
    p_max(j)=max(one_column_p);
end

%%
% ploting


v=v*gain;
i_max=i_max/gain;


figure
hold on
plot(v, i_max, 'k-')
grid on
set(gcf,'Position',[100 100 300 192])
set(gca,'FontName','Times')
set(gca,'FontSize',9)
title('IV characteristic');
xlabel('Total Voltage')
ylabel('Total Current')



figure
hold on
plot(v, p_max, 'k-')
grid on
set(gcf,'Position',[500 100 300 192])
set(gca,'FontName','Times')
set(gca,'FontSize',9)
xlabel('Total Voltage')
ylabel('Total Power')
title('Power vs Voltage')


figure
hold on
plot(i_max, p_max, 'k-')
grid on
set(gcf,'Position',[900 100 300 192])
set(gca,'FontName','Times')
set(gca,'FontSize',9)
xlabel('Total Current')
ylabel('Total Power')
title('Power vs Current')



if ti > 1
```

```
figure
hold on
plot(1:1:ti, max_p_in_row, 'k-')
grid on
set(gcf,'Position',[500 500 300 192])
set(gca,'FontName','Times')
set(gca,'FontSize',9)
title('Max Power vs Time');
xlabel('Time')
ylabel('Max Power')


energy = trapz(1:1:ti,max_p_in_row)
end
```

Set Parameters

```
function [ parameters ] = SetParameters...
    (resistance, gain, numPanels, sysInsol, Isc, Voc, numCells, A, Rs,
Rsh)

T = 273 + 25;
Vth = T*1.38e-23/1.6e-19;


%modification for Rs
Rs=Rs+(resistance/(gain^2));

Is1 = 0; %0.5*Isc/(exp((Voc/ncells)/Vth)-1);
Is2 = Isc/(exp((Voc/numCells)/(A*Vth))-1); % there is no 0.5 here

parameters = zeros(numPanels, 8);

for j = 1:numPanels
    parameters(j, 1) = sysInsol(j)* Isc;
end

parameters(:, 2) = Is1;% [A], I_0 is both Is1 and Is2
parameters(:, 3) = Is2;% [A],
parameters(:, 4) = A;% (normally 2), Ideality factor
parameters(:, 5) = Rs;% [ohms] =>  use in HQ's code
parameters(:, 6) = Rsh;% [ohms]
parameters(:, 7) = T;% [K]
parameters(:, 8) = numCells;
```

Series Calculation

```
function [ current ] = SeriesCalculations(vExt, numSeries, parameters)
% Series connection of identical panels, different insolation
% Using Newton-Raphson method taking Rs into account

numPVUsed = numSeries; % intially, all panels are used
parametersUsed = parameters;
[ current, voltage ] = NewtonPV(vExt, numPVUsed, parametersUsed);
```

```
% examine voltage for negative values
while min(voltage) < 0
    k = zeros(1, numPVUsed);
    ctr = 1;

    for j = 1:numPVUsed
        if voltage(j) < 0
            k(ctr) = j;
            ctr = ctr + 1;
        end
    end

    for j = numPVUsed:-1:1
        if k(j) > 0
            parametersUsed(k(j),:) = [];
            numPVUsed = numPVUsed-1;

            % if all the PVUsed has either negtive or NaN result so
they
            % all ignored return to the invoking function -1 to react
            if numPVUsed <= 0
                current = -1;
                return;
            end
        end
    end

    [ current, voltage ] = NewtonPV(vExt, numPVUsed, parametersUsed);
end
```

Newton PV

```
function [ current, voltage ] = NewtonPV( Vout, numPanels, parameters )
% Find current and voltage for series connected PV panels
% in a partially shaded PV array
% Use Newton-Raphon method
% by HQ, 05-01-2010
% Use new J matrix
% reduce size from 2n*2n to n*n
% for smaller size and fast matrix inverse
% by HQ, 06-07-2010
% THE PROBLEM IS: it converge very slow.
% Parameters
% Vout - external voltage at the termial of a PV array
% numPanels - # of panels
%   HACK!!! only count those panels NOT being bypassed
% params - parameter sets
%   HACK!!! only count those panels NOT being bypassed



global rows_index;

v = Vout / numPanels* ones(1, numPanels); % initial guess of cell
voltage
i = zeros(1, numPanels);
```

```
%dv = zeros(1, numPanels);
x = [v i];
% x contains solutions, in [v1, v2, ..., i1, i2, ...] format,
% NOTE, cells are in series connection

error = ones(size(v)); % errors
tol = 1e-3; % error tolerance

while max(abs(error)) > tol
    J = zeros(2 * numPanels, 2 * numPanels); % clear J at the beginning

    %%%%% forward path
    % find current
    for j = 1:numPanels

        % try to find the current if the initial guess of the search is
        % undefined just put a nigitive value so it will be ignored
        try
            x(j     +     numPanels)     =     CurrentCalculation(x(j),
parameters(j,:));
            if(~isfinite(x(j    +    numPanels))    ||    ~isreal(x(j    +
numPanels)))
                x(j) = -1;
                voltage = x(1:numPanels);
                i = x(numPanels + 1:2 * numPanels );
                current = i(numPanels);

                return;
            end
        catch error
            if(strcmp(error.identifier,
'MATLAB:fzero:ValueAtInitGuessComplexOrNotFinite'))
                x(j) = -1;
                voltage = x(1:numPanels);
                i = x(numPanels + 1:2 * numPanels );
                current = i(numPanels);

                return;
            end
        end
    end

    % forward path, find error
    % error(1) - error(ncells) are error of PV cells
    for j = 1:numPanels
        error(j) = Auxiliary(x(j+numPanels), parameters(j,:), x(j));
    end
    % error(ncells+1) - error(2*ncell-1) are error of series current
    for j = numPanels + 1:1:2 * numPanels - 1
        error(j) = x(j) - x(j + 1);
    end
    % voltage error
    %for j = ncells+2:2:2*ncells
        error(2 * numPanels) = sum(x(1:numPanels)) - Vout;
    %end
```

```
    %%%%% feedback path
    % calculate df/di and df/dv
    for j = 1:numPanels
        [J(j, j), J(j, j + numPanels)] = DifferentialCalculations...
            (x(j + numPanels), parameters(j,:), x(j));
    end
    % current-related J
    for j = 1:numPanels-1
        J(j + numPanels, j + numPanels) = 1;
        J(j + numPanels, j + 1 + numPanels) = -1;
    end
    J(2 * numPanels,1:numPanels) = 1;
    %J
    % fixed x
    %x = x - (inv(J) * error')';
    %if(rows_index > 130)

    %end
    x = x - (pinv(J)*(error'))';
   % if(rows_index > 130)
   %     x
   % end
end

voltage = x(1:numPanels);
i = x(numPanels + 1:2 * numPanels );
current = i(numPanels);
```

Current Calculation

```
function current = CurrentCalculation(voltage, parameters)
current = fzero( @(x) Auxiliary(x, parameters, voltage) , voltage );
```

Auxiliary

```
function f = Auxiliary(iTerminal, parameters, vTerminal)
% i is terminal current, now consider Rs
% vterminal is after series resistance,also PV cell voltage

Iph = parameters(1);
Is1 = parameters(2);
Is2 = parameters(3);
A = parameters(4);
Rs = parameters(5);
Rsh = parameters(6);
T = parameters(7);
ncells = parameters(8);


Vth = T*1.38e-23/1.6e-19; % Vth = T*K/q
vper = vTerminal + Rs*iTerminal; % PV cell voltage
% i1 = Is1*(exp(vper/(ncells*Vth))-1)
i1=0; %change 12/29


i2 = Is2*(exp(vper/(ncells*A*Vth))-1);
```

```
% end

i3 = vper/(ncells*Rsh);
% i3 = vper/(Rsh) nisha

current = Iph - (i1+i2+i3);

%current = photovoltaic(v, parameters);
f = current - iTerminal;
```

Differential Calculation

```
function [dv di] = DifferentialCalculations(i, params, vterminal)
% i is terminal current, now consider Rs
% vterminal is after series resistance,  also PV cell voltage

%Iph = params(1);
Is1 = params(2);
Is2 = params(3);
A = params(4);
Rs = params(5);
Rsh = params(6);
T = params(7);
ncells = params(8);

Vth = T*1.38e-23/1.6e-19; % Vth = T*K/q
vper = (vterminal + Rs*i)/ncells; % PV cell voltage
e1 = exp(vper/(Vth));
e2 = exp(vper/(A*Vth));

di = -Is1*Rs/Vth * e1 - Is2*Rs/(A*Vth) * e2 - Rs/Rsh - 1;
dv = -Is1/Vth * e1 - Is2/(A*Vth) * e2 - 1/Rsh;

end
```

REFERENCES

[1] R. A. Messenger and J. Ventre, *Photovoltaic Systems Engineering*, 3 ed.: CRC Press, 2010.

[2] (2012, Solar Energy Statistics. Available: http://www.statisticbrain.com/solar-energy-statistics/ (accessed on 03/19/2013)

[3] Renewable energy. Available: http://www.bp.com/subsection.do?categoryId=9037155&contentId=7068627 (accessed on 03/19/2013)

[4] M. H. Rashid, *Power electronics handbook devices circuits and applications*: Elsevier, 2011.

[5] D. Sera, R. Teodorescu, and P. Rodriguez, "PV panel model based on datasheet values " presented at the Industrial Electronics, 2007. ISIE 2007. IEEE International Symposium on, June 2007.

[6] A. Jenifer, N. R. Newlin, G. Rohini, and V. Jamuna, "Development of Matlab Simulink model for photovoltaic arrays " presented at the Computing, Electronics and Electrical Technologies (ICCEET), 2012 International Conference on, March 2012.

[7] T. Salmi, M. Bouzguenda, A. Gastli, and A. Masmoudi, "MATLAB/Simulink Based Modelling of Solar Photovoltaic Cell," *INTERNATIONAL JOURNAL of RENEWABLE ENERGY RESEARCH*, vol. 2, 2012.

[8] L. Gao, R. A. Dougal, L. Shengy, and A. P. Iotova, "Parallel-Connected Solar PV System to Address Partial and Rapidly Fluctuating Shadow Conditions " *Industrial Electronics, IEEE Transactions on*, vol. 56, pp. 1548 - 1556 2009.

[9] P. Roopa, S. E. Rajan, and R. P. Vengatesh, "Performance analysis of PV module connected in various configurations under uniform and non-uniform solar radiation conditions ", 2011.

[10] T. Esram and P. L. Chapman, "Comparison of Photovoltaic Array Maximum Power Point Tracking Techniques " *Energy Conversion, IEEE Transactions on*, vol. 22, pp. 439 - 449 2007.

[11] J. Young-Hyok, J. Doo-Yong, K. Jun-Gu, K. Jae-Hyung, L. Tae-Won, and W. Chung-Yuen, "A Real Maximum Power Point Tracking Method for Mismatching Compensation in PV Array under Partially Shaded Conditions," *Power Electronics, IEEE Transactions on*, vol. 26, pp. 1001 - 1009 2011

[12] M. Miyatake, M. Veerachary, F. Toriumi, N.Fujii, and H. Ko, "Maximum Power Point Tracking of Multiple Photovoltaic Arrays: A PSO Approach," *Aerospace and Electronic Systems, IEEE Transactions on,* vol. 47, pp. 367 - 380 2011.

[13] K. Kobayashi, I. Takano, and Y. Sawada, "A Study on a Two Stage Maximum Power Point Tracking Control of a Photovoltaic System under Partially Shaded Insolation Conditions," presented at the Power Engineering Society General Meeting, 2003, IEEE, July 2003.

[14] N. xiaohua. (2012, A New Improved Perturbation and Observation MPPT Control Algorithm in Photovoltaic System. 877-883.

[15] P. Petit and M. Aillerie, "Integration of individual DC/DC converters in a renewable energy distributed architecture " presented at the Industrial Technology (ICIT), 2012 IEEE International Conference on, March 2012.

[16] S. V. Dhople, R. Bell, J. Ehlmann, A. Davoudi, and P. L. Chapman, "A global maximum power point tracking method for PV module integrated converters " presented at the Energy Conversion Congress and Exposition (ECCE), 2012 IEEE, Sept. 2012.

[17] G. Petrone, G. Spagnuolo, and M. Vitelli. Distributed Maximum Power Point Tracking: Challenges and Commercial Solutions.

[18] P. L. Chapman. (2012, Embedded Electronics for the Solar Power Industry.

[19] F. Alfaris, "STOCHASTIC MODEL FOR SOLAR SENSOR ARRAY DATA," Master of Science, Electrical Engineering, Missour University of Science and Technology, 2012.

[20] B. A. Yount, "MANAGING A SOLAR SENSOR ARRAY PROJECT: ANALYZING INSOLATION & MOTIVATION," Master of Science, Engineering Managment, Missour University of Science and Technology, 2011.

[21] *SolarMade Company*. Available: http://www.solarmade.com/. (accessed on 03/19/2013)

[22] *Tern Inc.* Available: http://www.tern.com/portal/(accessed on 03/19/2013)

[23] *MATLAB GUI*. Available: http://www.mathworks.com/discovery/matlab-gui.html(accessed on 03/19/2013)

[24] N. Nagrajan, "Development of a graphical user interface for the study of parallel-connected solar array," Master of Science, Electrical Engineering, Missour University of Science and Technology, 2011.

[25] M. Jordan and J. Kimball, "Practical Performance Analysis of Complex Switched-Capacitor Converters " *Power Electronics, IEEE Transactions on,* vol. 26, pp. 127 - 136 2011.

VITA

Majed Meshal Al Abbass was born on October 25, 1986 in Kuwait. He completed his schooling in Asharq private school, Al Khafji, Saudi Arabia. Majed received his Bachelor of Engineering (B.E.) degree in Electrical and Control from the King Saud University, Riyadh, Saudi Arabia in May 2009. Later, he started working in Al Jouf University, Al Jouf, Saudi Arabia as a teaching assistant. He started his Master of Science program in Electrical Engineering at Missouri University of Science and Technology in August 2011. He received his Master degree in May 2013.